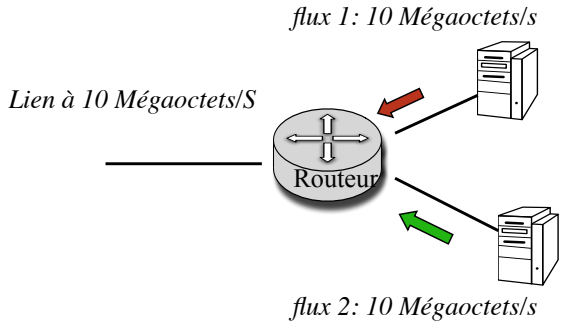
Le contrôle de congestion dans Internet

Introduction

- **Le problème**
  - But: Eviter que le réseau n'achemine plus correctement les paquets à cause d'un embouteillage
  - Dans le cas d'un embouteillage dans Internet, les routeurs détruisent les paquets qu'ils ne peuvent plus mémoriser
  - On peut retrouver les problèmes de congestion au niveau liaison de donnée
  - La résolution des collisions d'Ethernet est un contrôle de congestion

Exemple



- **Que se passe t-il dans le routeur ?**

Introduction

- **Différentes politiques possibles:**
  1. Se débrouiller pour que ça n'arrive jamais, impossible même sur les autoroutes il y a des carrefours
  2. Prévenir l'embouteillage par un contrôle des accès
  3. Attendre l'embouteillage et résoudre le problème une fois qu'il est là
     - » Méthode:
       - Surveiller le système
       - Envoyer des informations aux endroits où les décisions doivent être prise (péage des autoroutes)
       - Ajuster le comportement du système pour corriger le problème (agents aux carrefours, interdire certains accès)
- **Le contrôle de congestion peut être effectué à différents niveaux (Liaison, Réseaux, Transport)**

Contrôle de congestion au niveau liaison de donnée

- Pas de noeud intermédiaire
- Peut être nécessaire seulement pour les réseaux à diffusion
- Algorithme de résolution des collisions d'Ethernet (même problème pour le WIFI)
  - » Contrôle de congestion après coup
    - » Dans Ethernet l'émission est faite dès que possible et l'arrêt d'émission décidé au moment de la détection de la collision
  - » Peu efficace à forte charge (voir efficacité d'Ethernet)
  - » Dépend de la taille des paquets

Contrôle de congestion au niveau liaison de donnée

Commutateurs Ethernet actuels:

- Plus de collisions (full-duplex)
- Le commutateur ne re-émet que vers la destination après mémorisation du paquet grâce à une table de commutation (obtenue grâce aux premiers paquets)
- Possibilités de saturation de la mémoire du commutateur
- Des trames perdues implique une reprise des pertes par la couche transport; l'efficacité s'effondre
  - Paquets particulier dit de "PAUSE" vers la ou les machines source quand le commutateur s'approche de la saturation.
  - Les émetteur arrête d'émettre pendant la durée donnée dans le paquet PAUSE
- Fait aujourd'hui partie de la norme Ethernet

Le contrôle de congestion au niveau de la couche Réseau

- **Une couche réseau orientée connexion ou "datagram" ?**

Avantages d'une connexion:

- A l'ouverture de la connexion:
  - Réservation dans les routeurs de la mémoire nécessaire
- On évite la congestion en refusant des connexions
- Possibilités de gestion de QoS
- Décision de la route à suivre faite une seule fois

Inconvénients d'une connexion:

- Peu dynamique
- Problème des reprises de pannes
  - paquets déjà partis sur la route sont perdus
  - réouverture de connexion
  - reprise des pertes
- Les ressources sont allouées qu'on les utilise ou non

- **"Datagram":**

Avantages:

- Chaque paquet indépendant
- Routes différentes possibles
- Dynamique (exemple RIP),
- Possibilité de métrique dynamique (charge des routeurs) pour le choix des routes (dans la pratique n'existe pas)

Inconvénients:

- Déséquencement
- Calcul des routes et consultation de la table de routage pour chaque paquet donc charge CPU des routeurs importante
- Pas de réservation possible dans les routeurs pour une communication donnée
- Contrôle de congestion délicat

Solutions du contrôle de congestion dans Internet

- Contrôle de congestion non prévu dans les protocoles initiaux d'Internet
- Le protocole IP : Mode Datagram
- Difficile de réserver les ressources dans les routeurs à grande échelle (voir la gestion des QoS: protocole RSVP)
- Difficile de prévenir les congestions

➡ On essaye de limiter le travail des routeurs

➡ Le contrôle de congestion est résolu au niveau transport (TCP) !

Pour alléger le travail des routeurs: Fragmentation à la source

- Possibilité de fragmentation dans les routeurs
- Augmente beaucoup leur charge de calcul
- Re-assemblage de toute façon seulement à l'arrivée (les fragments peuvent suivre des routes différentes)
- Si un fragment se perd il faut tout re-émettre
- Estimation du MTU(Maximum Transmission Unit) par la source:
  - grâce à l'émission d 'un paquet ICMP (comportant la taille max) par un routeur vers la source lorsque le paquet ne peut être fragmenté (flag Don 't fragment)
  - A la réception du paquet ICMP la source (TCP) change la valeur du MTU
  - Fonctionne bien si les routes sont stables

Optimisation du travail des routeurs: Association connexion Transport et route

Implémenté la plupart du temps aujourd'hui dans les routeurs

On s'approche d'une solution avec circuit virtuel

- Association de la route à l'identification de la connexion de niveau transport (UDP ou TCP) : @internet et Numéro port source, @internet et Numéro port destination
- Fait pour le premier paquet, pour les suivants économie de la consultation de la table de routage(sauf modification dynamique de la table)
- Arrêt de l'association par timer d'inactivité

Limiter l'émission de petits paquets

Déblocage dans le contrôle de flux («fenêtre stupide»)

- Cas d'une petite libération du buffer de réception
- Passage du champ WIN de 0 à *x* petit
- Emetteur ne peut envoyer que *x*
- Exemple: l'application récupère les données octet par octet (talk)
- 58 octet d'entête TCP/IP/Ethernet pour un octet de donnée
- Le récepteur ne débloque l'émetteur que lorsqu'il a libéré une quantité importante du buffer
- En pratique minimum entre taille maximale des segments et moitié du buffer de réception

Retard des acquittements de TCP

- Un acquittement: 58 octets d'entête
- On essaye de profiter des données pour émettre les acquittements
- Le récepteur n'émet pas l'acquittement immédiatement
- Il attend un temps fixe (en pratique 200ms, configurable sous *bsd*) en espérant
  - soit profiter d'une donnée circulant dans le sens opposé pour « transporter » l'acquittement
  - soit d'acquitter plusieurs paquets à la fois (acquittements « cumulatifs »)
- Exemple: dans l'application Telnet
  - l'écho des caractères tapés et l'acquittement sont dans le même paquet
- **Attention** cela peut diminuer l'efficacité de la récupération d'erreur

Délais de propagation variables

- Idée: regrouper les petits paquets donnés par l'application dans un même paquet TCP
- Mais cela peut diminuer la réactivité de l'application
- Dommage si le réseau est "petit"
- TCP tourne aussi bien entre :
  - deux machines proches (même réseau local) : temps de propagation de l'ordre de 25 microsecondes
  - deux machines éloignées avec de nombreux routeurs intermédiaires: temps de propagation de l'ordre de la milliseconde, voir de la seconde
- Idée: Ne pas émettre des petits paquets si l'on est sur un réseau étendu

Algorithme de Nagle

- TCP attend que sa file d'attente se remplisse avant d'émettre si le temps d'aller retour (RTT: Round Trip Time) est grand
- On évite ainsi la saturation des routeurs intermédiaires par des petits paquets
- Mise en oeuvre:
  - TCP ne peut émettre qu'un paquet de petite taille sans avoir d'acquittement
  - Si le temps de propagation est grand, il attend d'autant et du coup sa file d'émission se remplit

Contrôle de congestion au niveau transport

- **Parce que peu de chose sont réalisables au niveau IP , on limite « la casse » dans TCP**
  - Mauvaise solution car la couche transport ne voit que l'entité destinataire, elle n'a pas de connaissance du chemin et des routeurs intermédiaires
  - On peut limiter les congestions en contrôlant les flux de sortie sur les réseau chez l'émetteur
  - Le contrôle de congestion peut alors être traité comme un contrôle de flux ou l'entité réceptrice serait le réseau

Contrôle de congestion au niveau transport

Dans le protocole TCP plusieurs solutions ont été élaborées

1- Quand le réseau est transparent, c'est à dire qu'aucune information explicite provient du réseau

- Solution de résolution (et non de prévention)
- Le réseau passe dans un état de congestion, TCP réagit en diminuant les débits des émetteurs
- Cela implique de nombreuses pertes de paquet

2- **Le réseau envoie des informations de risque de congestion aux émetteurs**

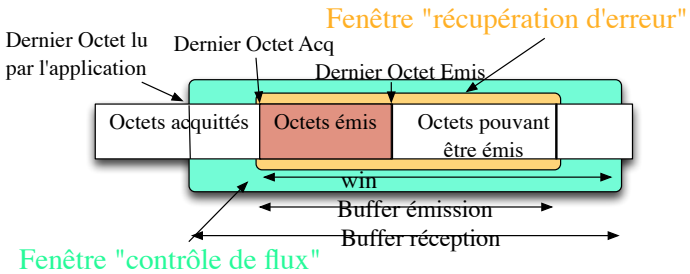
-Solution de prévention

-Quasiment plus de pertes de paquets

Détection de congestion ?

- **Si les routeurs n'envoient pas d'information concernant les congestions comment un émetteur peut-il détecter une congestion ?**
  - La plupart du temps la fiabilité des lignes (niveau physique) est très grande et donc très peu de pertes de ce côté là
  - Perte de paquet: destruction par un des routeurs qui arrive à saturation
  - Donc une perte est signe de congestion
  - Dans TCP une perte est détecté par le contrôle d'erreur
    - Un timer de retransmission se déclenche

Rappel : les fenêtres à anticipation



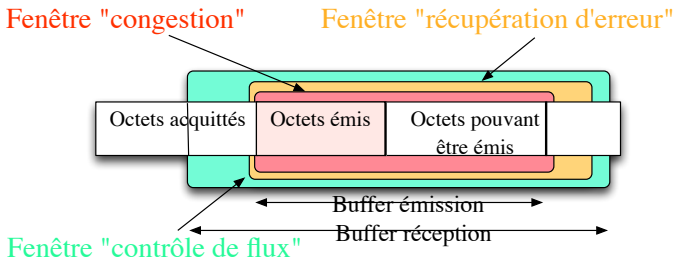
- Les deux fenêtres avancent au fur et à mesure de l'arrivée des acquittements
- Celle de la récupération d'erreur au vu du Numéro d'acquittement
- Celle du contrôle de flux au vu du Numéro d'acquittement et du champ Window
- L'émetteur se bloque sur la fenêtre la moins «avancée»

La fenêtre de congestion

- Contrôle de congestion traité comme un contrôle de flux dont l'entité réceptrice à ne pas saturer serait alors « le réseau »
- Idée : rajouter au contrôle de flux et la récupération d'erreur de TCP, une troisième fenêtre dite fenêtre de congestion (appelé ***cwnd***)
- La taille de la fenêtre de congestion va varier en fonction de l'estimation de la congestion du réseau

La fenêtre de congestion

- Du coup, le nombre d'octets pouvant être émis sans attendre d'acquittement est déterminé suivant l'"avancement" des trois fenêtres : congestion, contrôle de flux (win) et récupération d'erreur
- L'émetteur se bloque sur la fenêtre la moins «avancée»



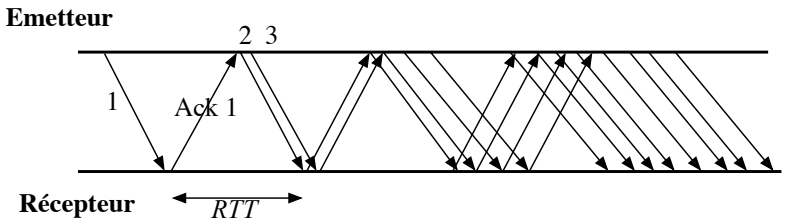
Principe de la fenêtre de congestion

- Comment fixer la taille de la fenêtre de congestion ?
- Découvrir la bande passante disponible le plus rapidement possible
- Comme on n'a aucune connaissance du réseau, l'idée est de démarrer d'une valeur très basse mais d'augmenter rapidement pour que l'efficacité reste bonne si le réseau est performant
- On augmente la taille de la fenêtre de congestion (donc le débit) jusqu'à que l'on arrive à une perte qui signifie la congestion
- L'émetteur va alors diminuer la taille de la fenêtre pour diminuer son débit d'émission

Réglage de la taille de la fenêtre de congestion

- Soit MSS la taille maximum des segments envoyés sur le réseau
- Algorithme dit du démarrage lent (***slow start***)
  - cwnd= MSS (un seul paquet)
  - A la réception d'un acquittement faire
    - cwnd= cwnd + MSS
  - Si un temporisateur de re-émission expire (perte donc congestion)
    - cwnd= MSS
- Le fait d'augmenter d'un MSS à chaque acquittement revient à doubler la taille de la fenêtre tous les RTT (Temps aller-retour)
- Le ***slow start*** commence très bas mais augmente exponentiellement
- La fenêtre de congestion s'arrête de grandir quand elle devient égale à la fenêtre de récupération d'erreur

Exemple Slow start



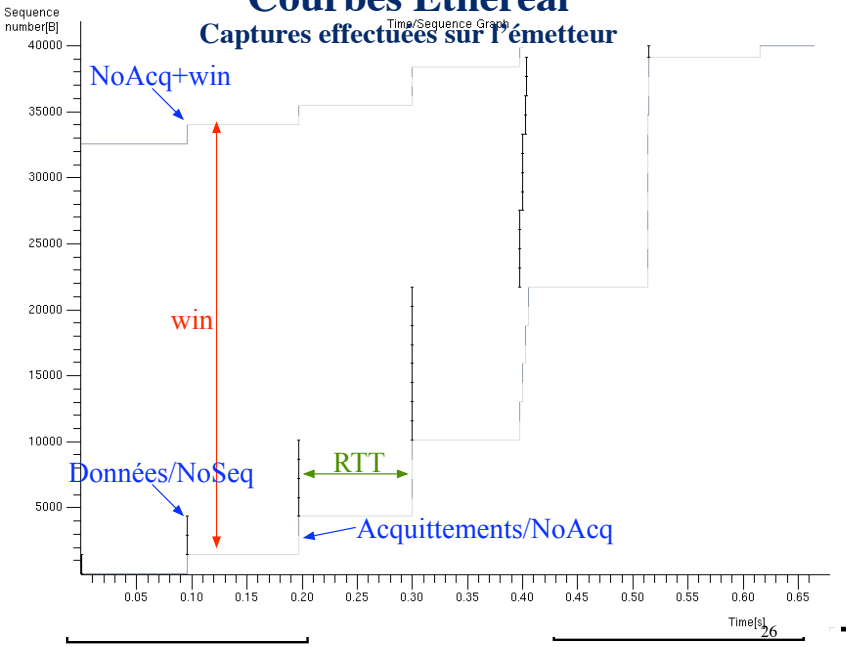
- Augmentation de la charge du réseau jusqu'à la congestion

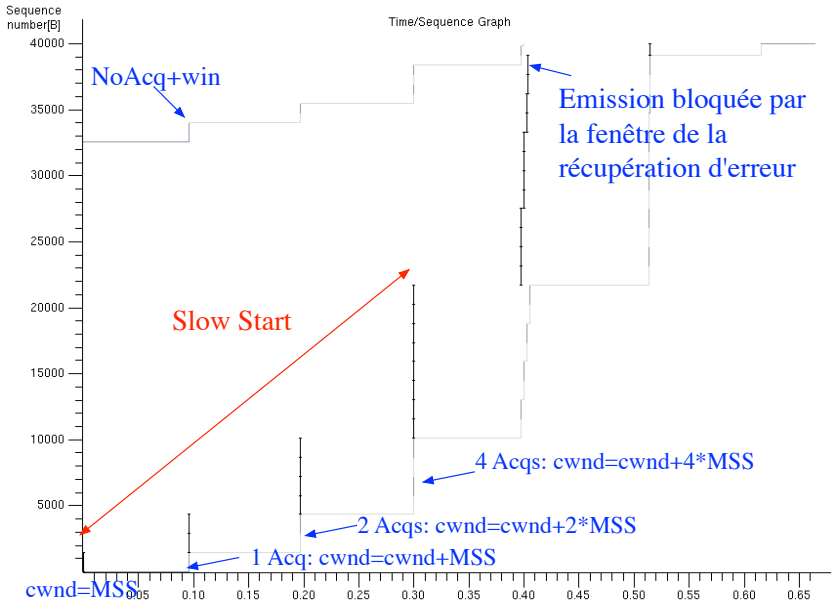
Cas de perte

- En cas de perte, on repart avec une fenêtre à 1 MSS (Certaines versions récentes de TCP démarrent à 3 MSS)
- Pour ne pas recommencer à arriver à une congestion, on définit un seuil appelé twnd au delà duquel la fenêtre sera augmentée linéairement (phase d'***évitement de congestion***):
  - cwnd= cwnd+1 tous les RTT
- Ce seuil fixé très grand au départ prend la valeur de la fenêtre divisé par 2 au moment de la perte
- Par tâtonnements successifs on va s'approcher du débit maximal sans qu'il y est de congestion
- A tout moment la charge du réseau change et les émetteurs s'adapte en conséquence

Courbes Ethereal

Captures effectuées sur l'émetteur

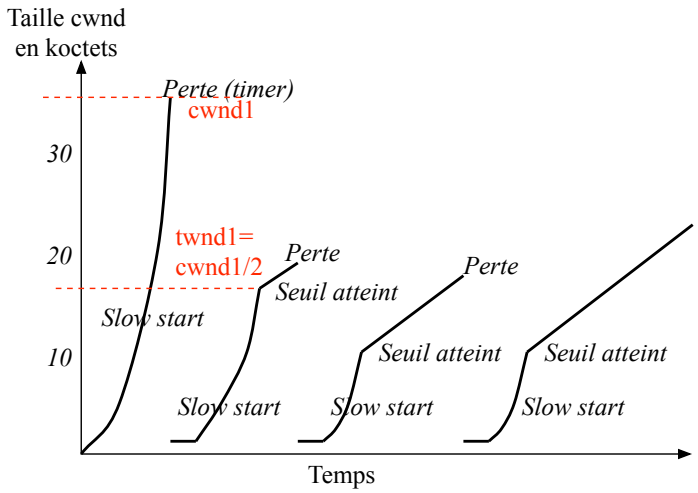




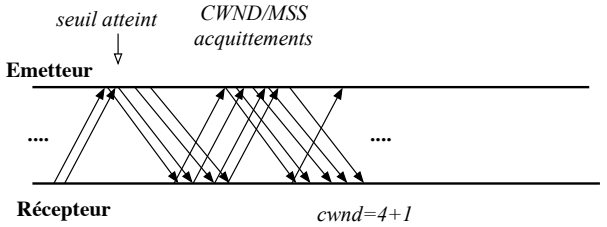
Algorithme Tahoe (1988)

- cwnd= MSS, twnd= 64 Kbytes
- A la réception d'un acquittement faire
  - si cwnd < twnd ("slow start") alors cwnd=cwnd + MSS
  - sinon ("évitement de congestion") cwnd= cwnd + MSS*(MSS/cwnd)
- Si un temporisateur de re-émission expire (perte donc congestion)
  - twnd = cwnd/2
  - cwnd= MSS
- Le seuil twnd est divisé par 2 en cas de congestion pour ne pas retourner en cas de congestion
- Il y a cwnd/MSS ACK qui reviennent tous les RTT, donc au bout d'un RTT:
  - cwnd = cwnd + (MSS*(MSS/cwnd))*cwnd/MSS = cwnd + MSS

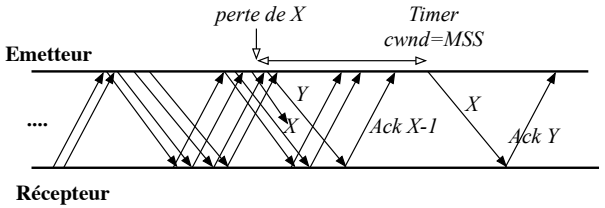
Exemple Tahoe



- **Phase d'évitement de congestion**

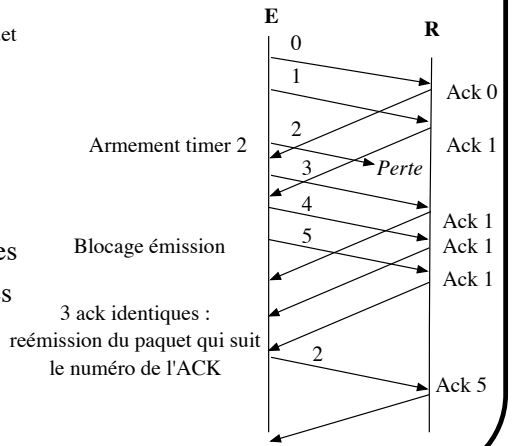


- **Cas de perte (expiration de timer)**



Amélioration sur ACK dupliqués

- Dans la récupération d'erreur de TCP il y a une optimisation :
  - En cas de perte TCP re-acquitte le dernier paquet bien arrivé.
  - En cas de 3 ACK dupliqués, il suppose que le paquet suivant n'est pas arrivé.
  - Il le re-émet tout de suite sans attendre la sonnerie du timer (plûtot surestimé)
- Idée: Différencier les pertes reconnues par timer et par acquittements dupliqués



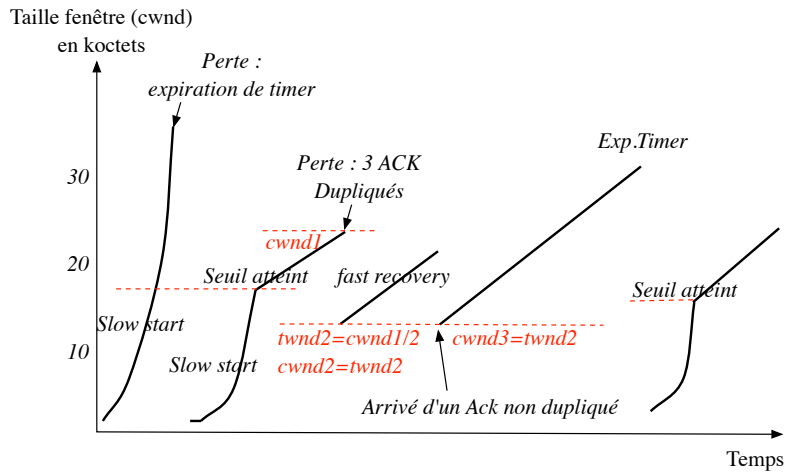
Amélioration- TCP RENO (1990)

- En effet si des ACK dupliqués arrivent c'est que d'autres paquets sont arrivés, la congestion n'est pas si dramatique
- Dans TCP RENO, en cas de trois Acks identiques:
  - twnd=cwnd/2, cwnd= twnd

Phase de Fast Recovery

- Après une reémission sur Ack dupliqués, on rentre en phase de récupération rapide (***fast recovery***)
  - Cette phase dure jusqu'à que soit
    - » l'on reçoive un acquittement non dupliqué (acquittement de la re-émission), on passe alors en phase d'évitement de congestion avec cwnd= twnd
    - » ou lorsque le timer de ré emission du paquet reémis se déclenche (dans ce cas on repart en slow start)
  - Pendant cette phase à la réception d'un Ack dupliqué, on continue d'augmenter la fenêtre de façon linéaire, des paquets pourront donc continuer à être envoyés. Cela permet de ne pas baisser le débit brutalement

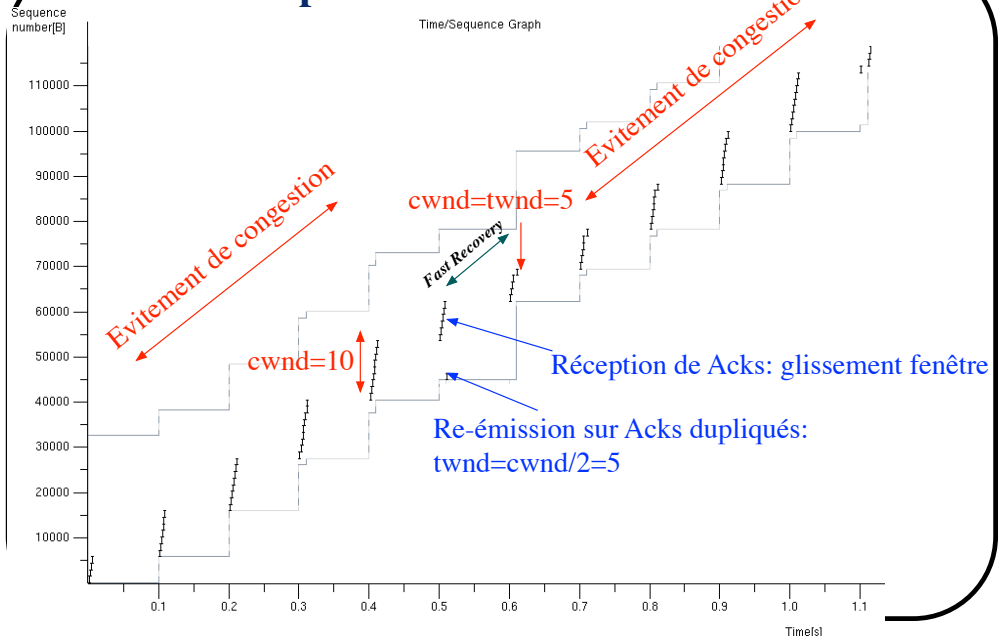
Exemple TCP RENO



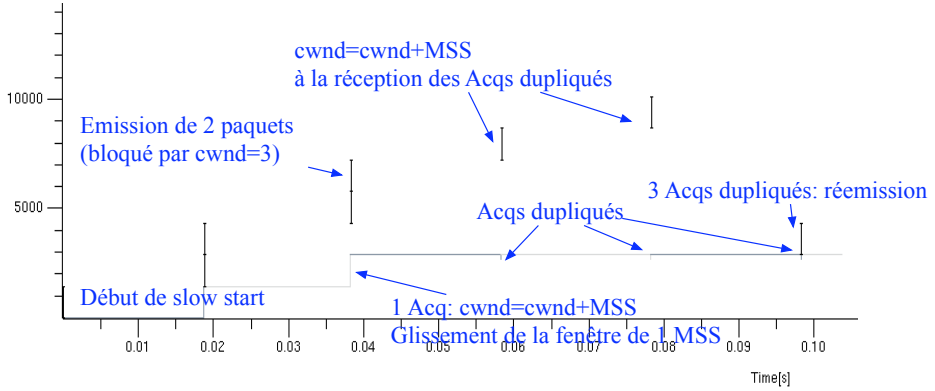
Algorithme RENO (1990)

- cwnd= MSS, twnd= 64 Kbytes
- A la réception d'un acquittement faire
  - si cwnd < twnd ("slow start") alors cwnd=cwnd + MSS
  - sinon ("phase d'évitement de congestion") cwnd= cwnd + MSS*(MSS/cwnd)
- Si un temporisateur de re-émission expire
  - twnd = cwnd/2
  - cwnd= MSS
- Si réception de 3 Acq identiques
  - On entre en phase de Fast Recovery
  - ✓ Si réception d'un Ack Dupliqué alors cwnd=cwnd + MSS
  - ✓ Si Réception d'un Ack non dupliqué alors
    - La phase de Fast Recovery est finie:
      - twnd= cwnd/2
      - cwnd= twnd
      - on repasse en phase d'évitement de congestion

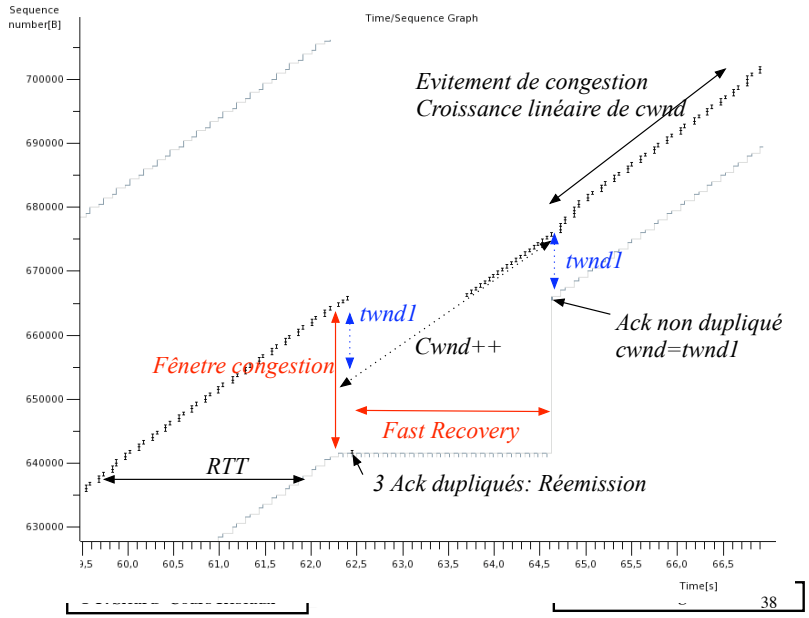
Exemple 1 - TCP RENO



Exemple 2- TCP RENO (capture sur l'émetteur)



Exemple 3 de capture TCP RENO



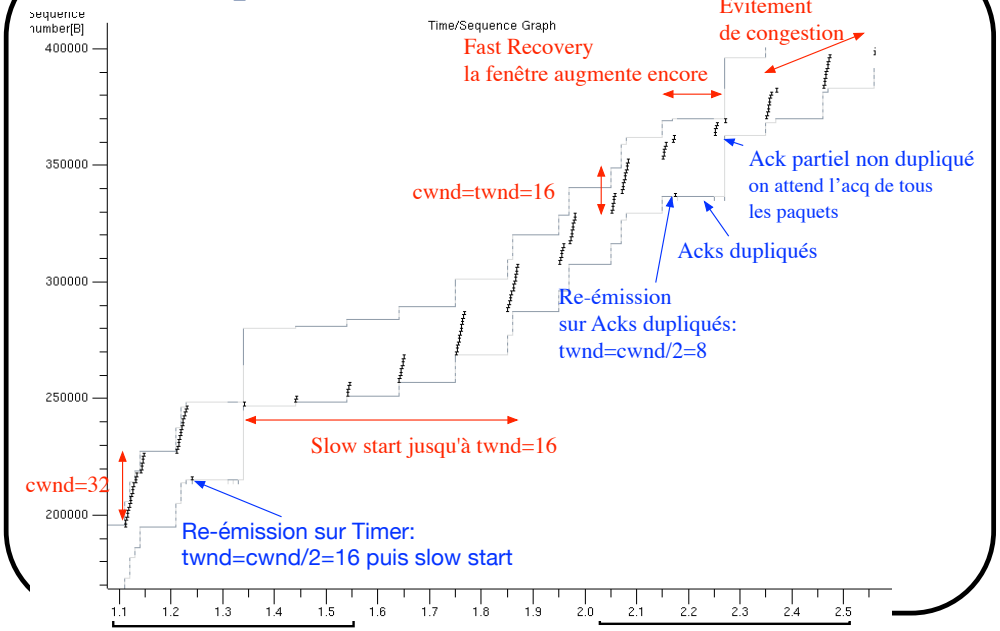
TCP NEW RENO (1999)

- Amélioration de Reno en cas de plusieurs pertes rapprochées
- Plusieurs pertes consécutives proviennent de la même congestion
- But: dans ce cas là ne pas trop baisser le débit
- Réaction de Reno en cas d'une rafale de pertes :
  - Dans la phase de fast recovery:
    - » Après la réception d'un ack non dupliqué (mais ***partiel***) Reno repasse en phase d'évitement de congestion avec une fenêtre divisée par 2
    - » Reno va rediviser par deux la fenêtre pour chaque perte suivante (si il y a encore des ACK dupliqués) ou même sûrement repartir en slow start suite à l'expiration d'un timer de reémission

TCP NEW RENO

- Dans ***New reno*** à la réception d'un ack non dupliqué :
  - Si c'est un ACK partiel (Ack non complet de tous les paquets déjà émis), on ne repasse pas en phase d'évitement de congestion et on bloque l'émission (contrairement à Reno)
  - C'est seulement à la réception de l'ACK de tous les paquets émis depuis le départ en Fast Recovery que l'on passe en phase d'évitement de congestion

Exemple - TCP New RENO



A trouver exemple avec 2 pertes consécutives

Autres améliorations

- **Vegas 1994**: consiste à ne plus attendre de perte pour ralentir son rythme d'émission
  - Pour cela la méthode se base sur l'évolution du RTT
  - Si le RTT augmente c'est que les files d'attentes dans les routeurs augmentent et que l'on s'approche de la congestion
  - On diminue alors la fenêtre jusqu'à que le RTT diminue

Adaptation à des réseaux de type particulier

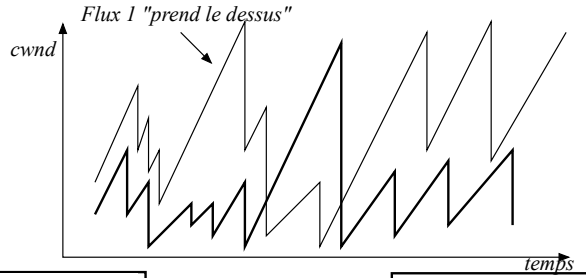
- **Réseaux sans fil:**
  - Westwood +, Veno
  - Pertes aléatoires non dues à des congestions
  - Se base sur l'évolution du RTT
  - Moins de baisse de débit en cas de perte que les autres méthodes
- **Réseaux haut débit et grand temps de propagation:**
  - Compound TCP (ou CTCP): Mélange de Reno et Vegas
  - HighSpeed TCP (HTCP, HSTCP), BIC (Binary Increase Congestion), CUBIC, TCP Illinois,…

Dans la pratique

- Le contrôle de congestion par défaut dépend du système d'exploitation utilisé
  - A partir de Free bsd 9.0 : New reno
  - Linux : CUBIC
  - Windows: Reno puis Compound TCP
- « plug-in » possible pour d'autres méthodes
- On peut le modifier à l'aide de paramètre système (***sysctl***) suivant par exemple le type de réseau utilisé
- Cela peut modifier de façon importante l'efficacité du « réseau »
- Le contrôle de congestion au niveau TCP est compliqué et reste un sujet d'actualité

Observation de flux TCP

- Le contrôle de congestion de TCP Reno et Tahoe aboutissent à des phénomènes d'oscillation (augmentations fortes puis diminutions fortes des flux)
- Dans le cas de plusieurs flux TCP, on observe des débits très variables suivant les flux
- En effet en cas de saturation d'un routeur
  - De nombreux paquets peuvent être supprimés (congestion forte)
  - Un flux peut être pénalisé plus qu'un autre



Le mécanisme RED (Random Early Detection)

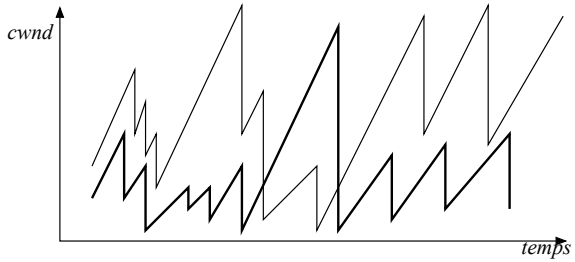
- L'idée est d'avertir les émetteurs que l'on approche d'une congestion avant qu'elle n'arrive
- Une façon de ralentir le rythme des émetteurs est de supprimer de façon aléatoire des paquets arrivant si le taux de remplissage a atteint un seuil donné (les flots à débits plus important ont plus de chance de perdre un paquet)
- La probabilité de suppression dépend du remplissage du tampon
- Deux seuils, ***tmin*** et ***tmax***, soit ***taille*** le remplissage du buffer
  - »si **0 ≤ taille < tmin**: le paquet passe
  - »si **tmin ≤ taille < tmax**: le paquet est rejeté avec une probabilité dépendante de la taille de la file
  - » si **tmax ≤ taille**: le paquet est rejeté

Le RED et TCP

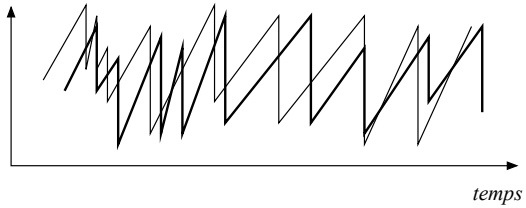
- Grâce au RED on aboutit à
  - Un lissage des flux, on évite vraiment les congestions
  - Une équité entre les flux TCP, la suppression étant aléatoire, les paquets des flux les plus importants ont plus de chance d'être détruits
- Le mécanisme RED est implémenté aujourd'hui dans la plupart des routeurs

Evolution de cwnd

- **Sans RED (2 flux TCP en concurrence)**



- **Avec RED**



La solution des paquets de prévention de congestion

- Pour faire encore mieux que le RED, il faudrait mieux plutôt que de détruire des paquets, prévenir (sans détruire) les émetteurs de l'approche de la congestion
- Amélioration définie en 1994 et utilisée couramment depuis 2001
- Mécanisme de **notification de congestion explicite** : ECN (Explicit Congestion Notification)

Principe de la notification de congestion

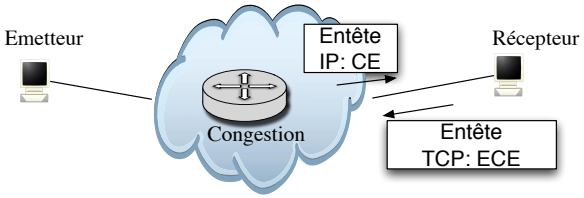
- **Les paquets d'engorgement: une solution pour prévenir la congestion quand elle n'est pas loin**
  - Chaque routeur surveille le pourcentage d'utilisation de ses interfaces de sorties qui sert à calculer une valeur à laquelle est associée un seuil de risque de congestion
  - Dès qu'il y a dépassement du seuil sur une sortie, le routeur va signaler à la source la congestion
  - Quand la source reçoit un paquet signalant un engorgement, elle réduit le débit d'émission vers l'adresse destination spécifiée
  - Mélange entre la couche réseau IP (notification dans les routeurs) et la couche transport TCP (limitation du débit des émetteurs)
  - Malheureusement peu utilisé dans les routeurs
  - Suivant les OS, implémenté ou pas, par défaut ou pas

Principe de l'ECN (Explicit Congestion Notification)

- TCP peut spécifier:
  - si il gère l'ECN : Option à l'ouverture de la connexion
  - Si il faut qu'un émetteur réduise son débit : bit ECE (Congestion Notification Echo) positionné dans les acquittements d'un flux de donnée
  - si il a réduit sa fenêtre de congestion : bit CWR : Congestion Window Reduce
- Les routeurs peuvent marquer dans l'entête IP les deux bits inutilisés du champ de "Differentiated Services" d'IP
  - 00: ECN non utilisé
  - 01 ou 10 : ECN utilisé
  - 11: ECN utilisé et congestion signalée : CE (Congestion Encountered),

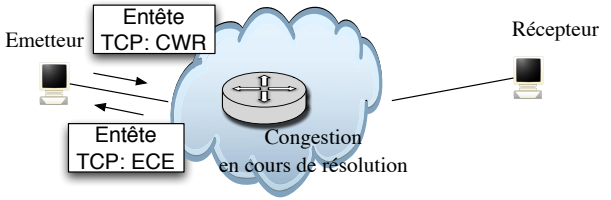
Principe de l'ECN

- Si l'émetteur est compatible ***ECN***, le routeur à l'approche d'une congestion (seuil de remplissage des files de sortie) marque le paquet (***CE*** bit)
- A la réception d'un paquet marqué ***CE*** (IP)***,*** le récepteur place un drapeau dans le flux inverse (bit ***ECE*** dans entête TCP)



Principe de l'ECN

- L'émetteur à la réception d'un ***ECE*** réduit la fenêtre de congestion
- Il marque alors les paquets en ***CWR*** dans l'entête TCP pour signaler qu'il a ralenti son rythme d'émission
- A la réception d'un paquet marqué ***CWR***, le récepteur arrête de marquer les paquets par ***ECE***



En conclusion

- TCP gère les congestions par diverses solutions:
  1. Diminution du débit de l'émetteur après un début de congestion: nombreuse perte de paquets
  2. Amélioration par RED : Diminution du débit de l'émetteur à l'approche de la congestion mais encore des pertes
  3. Amélioration ECN: Diminution du débit de l'émetteur à l'approche de la congestion mais plus de perte
- Concurrence de flux TCP:
  - Grâce au contrôle de congestion TCP est naturellement équitable
  - Pour N flux arrivant sur un routeur, chacun utilise 1/N de la bande passante possible (à essayer en TP)
- Concurrence de flux TCP-UDP:
  - Non équitable, UDP ne se soucie pas des congestions
  - Il existe DDCP (Datagram Control Congestion Protocol) au dessus de UDP permettant d'utiliser le mécanisme ECN