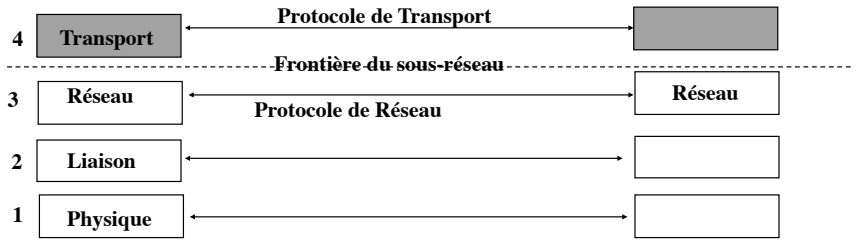
La couche Transport



Services de la couche transport

- Transfert fiable efficace, sûr et économique d'informations de bout en bout
- Dans l'idéal, répondant à des Qualités de Services (QoS) diverses
- Dernière couche avant les "applications"
- Les qualités de services nécessaires aux applications
  - Très variables
  - Exemples:
    - Transfert de fichier: Taux erreur nul, débit n'est pas primordial, le temps de transit non plus
    - Téléphone: Taux d'erreur peut être non nul, débit minimum indispensable, latence maximale (< 0,25 s)

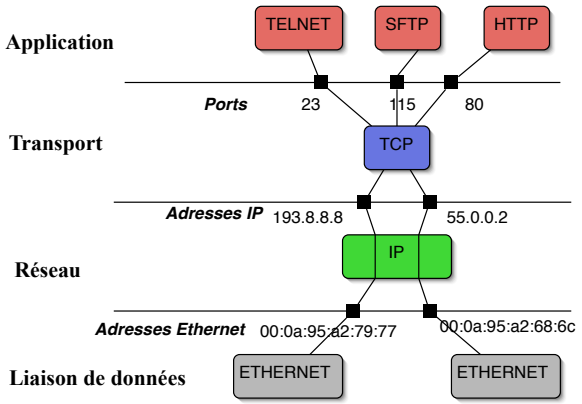
Les services de la couche transport d'Internet

- Rappel: IP sans connexion non fiable, déséquencement possible, délai de transfert très variable
- Protocoles pensés au départ pour le transfert de fichier et l'utilisation de machine à distance
- Deux types de services suivant les besoins de l'application à développer
  - Sans connexion, aucune QoS: User Datagram Protocol (UDP)
  - Avec connexion: Transport Control Protocol (TCP)
    - » Ouverture et fermeture de connexion
    - » Fragmentation et reassemblage
    - » Contrôle de flux et récupération des erreurs
- TCP et UDP se sont imposés naturellement. Les normes OSI ont été définies en parallèle mais elles sont arrivées trop tard
- TCP garantie seulement un taux de perte nul
- Dans les normes OSI, il existe différents niveaux de QoS pouvant être utilisés suivant le réseau sous-jacent

Fonctionnalités des protocoles transport d'Internet

- **Protocoles client/serveur:**
  - » Le serveur se met en attente de demandes
  - » Le client initie le dialogue par une demande
- Interface des "**sockets**": librairies de primitives d'accès aux protocoles transport TCP et UDP
- 1 serveur : 1 **numéro de port fixé**
  - Côté serveur: réservés pour applications standards
    - Fichier /etc/services
    - Exemple: HTTP port 80 en TCP
  - Côté client: alloués dynamiquement

Multiplexage vers les applications



- Connexion définie par les Adresses Internet source et destination et les numéros de ports source et destination
- Informations se trouvant dans les entêtes réseau (IP) et transport (TCP ou UDP)

Les services du protocole UDP

- Mode sans connexion (Datagram)
- Multiplexage vers les applications
- Aucun contrôle de flux et de récupération d'erreur
- **Entête UDP:**
  - Numéros de port (sur 2 octets)
  - Longueur en nombre d'octets (sur 2 octets)
  - Détection d'erreur optionnelle (par "checksum")

Port source	**Port destination**
Longueur	**Détection d'erreur**
Données UDP	

Services du protocole TCP

- Service orienté connexion
  - Ouverture et libération de la connexion
- Segmentation et re-assemblage des messages
  - Pas de notion de paquets de donnée à la réception : flux d'octets
- Rétablir l'ordre des paquets
  - Numérotation des octets de données
- Multiplexage vers plusieurs applications (numéro de port)
- Transfert de données exprès
- Contrôle de flux
- Détection des paquets erronés ou perdus
- Récupération des erreurs par re-émission
- Détection d'inactivité

Le format du paquet TCP

Port source			**Port destination**
Numéro de séquence			
Numéro d'acquittement			
Lg de l'entête		**Flags**	**Fenêtre**
Détection d'erreur			**Pointeur**
Options			

- **Flags: Urgent, Ack, Psh, Rst, Syn, Fin**

Connexion de niveau Transport

• Trois phases de communication :

1- établissement ou ouverture de la connexion

- permet de savoir si l'entité distante est prête
- échange possible des paramètres de connexion

2- la communication proprement dite (échange des données)

3- la rupture ou fermeture de la connexion

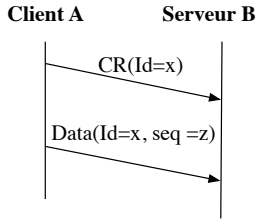
• Facilite certains services comme le contrôle de flux et la récupération des erreurs

L'établissement d'une connexion

- **Problème :**
  - Etablir une connexion virtuelle entre deux entités
  - Une fois la connexion établie, l'échange des paquets de données est bidirectionnel entre les deux entités
  - Comment définir une connexion sans ambiguïté ?
  - Comment associer une donnée à une connexion ?
- **Rappel**
  - Le niveau transport dans Internet est mis en œuvre au dessus d'une couche réseau non fiable
    - Perte de paquet
    - Retardement de paquets
    - Deséquencement des paquets

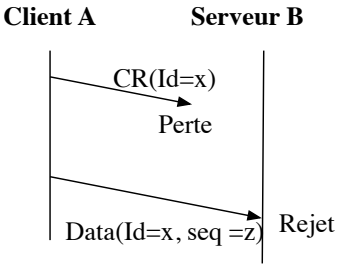
Mécanisme d'établissement de connexion

- Envoi d'un paquet spécial pour l'ouverture de la connexion
  - Connexion Request (CR)
- Identificateur de connexion dans l'entête des paquets de demande de connexion et des données appartenant à la connexion
- Numéro de séquence pour les données pour différencier les paquets d'une même connexion
- Chaque paquet de donnée circulant est identifiée de façon unique par un couple : <identificateur de connexion, n°SéquenceTPDU>



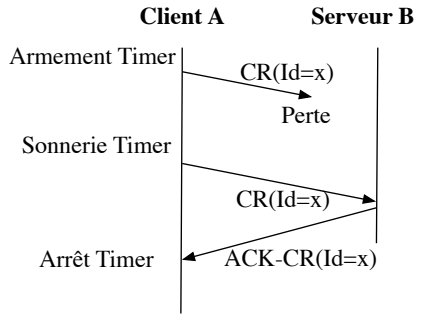
Perte de la demande de connexion

- Les données sont envoyées mais rejetées en B, pour lui il n'existe pas de connexion



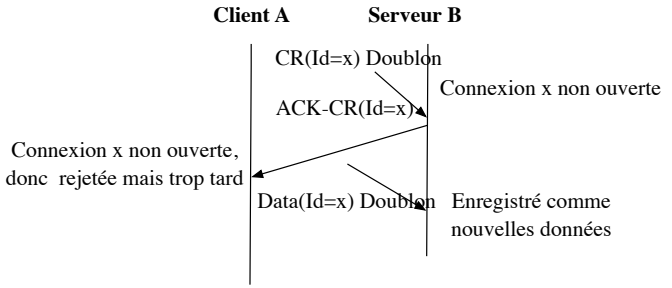
Perte de la demande de connexion

- Donc on rajoute un paquet d'acquittement (ACK-CR) permettant à A de s'assurer que la demande de connexion a bien été reçu par B
- On ajoute un mécanisme de timer et de re-émission de la demande de connexion en cas de non réception de l'acquittement



Problème des doublons

- Re-émission possible d'un CR alors que le CR initial a simplement été retardé et non perdu
- Re-émission possible de données dans une connexion précédente

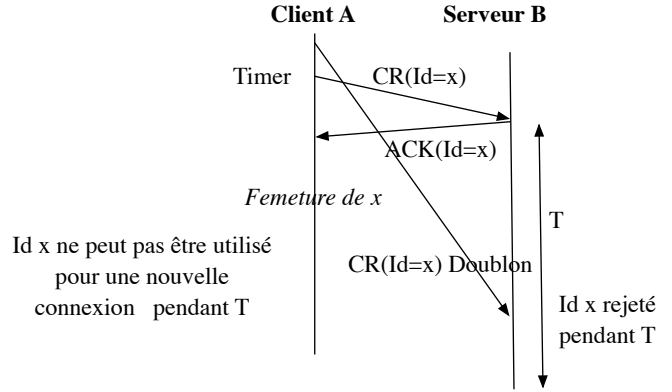


Solution au problème des doublons

- **Se rappeler quels identificateurs de connexion ont déjà été utilisés**
  - Pour reconnaître les doublons de demandes d'ouverture de connexion, on conserve les références des connexions fermées
  - Si on veut résister aux pannes: sauvegarde sur disque dur des identificateurs déjà utilisés
- **Problème : le nombre d'identificateurs n'est pas infini**
  - Il faudra reprendre à un moment les identificateurs déjà utilisés
  - On se débrouille pour avoir une durée de vie limitée des paquets dans le réseau
  - Soit T la durée de vie maximale d'un paquet sur le réseau

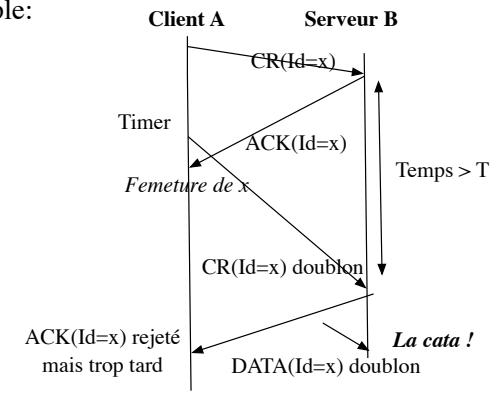
Solution en deux étapes

- Première idée : on garantie que l'on ne réutilise pas un identificateur avant un temps T
- Par exemple en prenant une plage de numéro «assez» grande



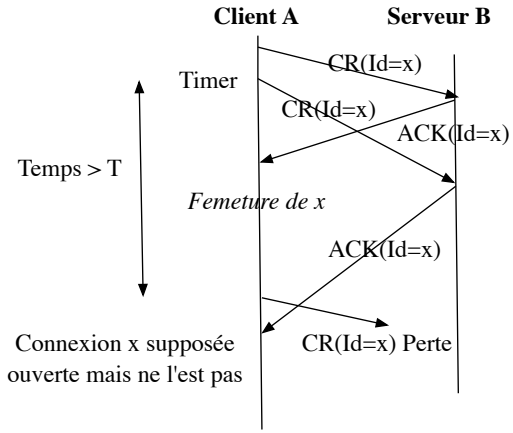
Mise en défaut de la solution à deux étapes

- Doublon de CR et de donnée
- Le problème est que le temps de réponse n'est pas borné
- Exemple:



Mise en défaut de la solution à deux étapes

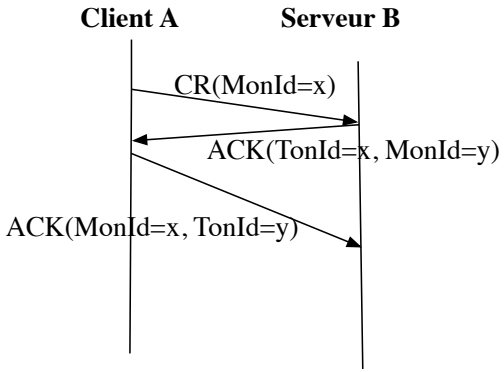
- Ambiguité sur l'ouverture de la connexion
- Exemple:



Solution avec durée de vie limitée(2)

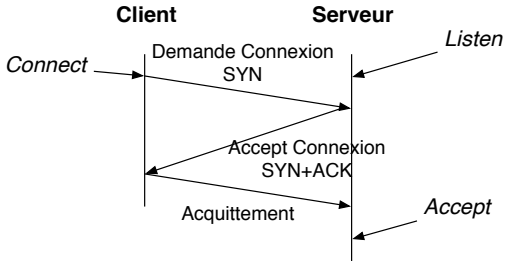
- **Une méthode d'ouverture de connexion fiable (Tomlinson 1975):**
  - Une connexion est identifiée par deux références uniques choisies respectivement par chacune des entités de transport
  - L'identification de la connexion est établie au cours d'un protocole d'ouverture à trois phases :
    - » A envoie une demande de connexion : CR (mon_id = x)
    - » B répond par une acceptation de connexion: AC (mon_id = y, ton_id = x)
    - » A envoie un acquittement: ACK(mon_id = x, ton_id = y)
- **Si on garantit l'unicité de x et y sur T unités de temps (indépendamment de chaque côté) alors**
  - on peut garantir que les doublons seront rejetés
  - il n'y aura pas d'ambiguité sur les connexions (même en présence de panne)

Exemple d'ouverture à trois étapes

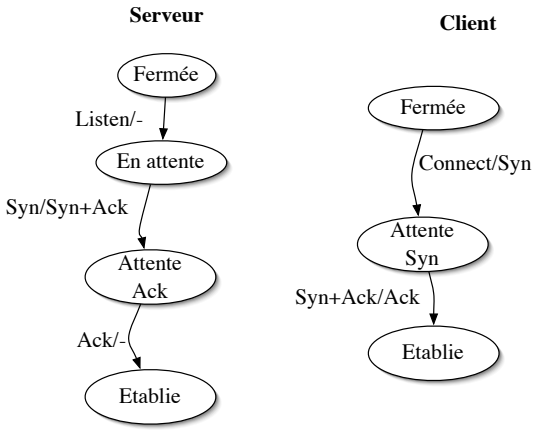


Etablissement d'une connexion TCP

- **Des paquets particuliers pour ouvrir la connexion**
  - Demande de connexion: Flag SYN=1 et ACK =0
  - Acceptation de connexion : Flags SYN= 1 et ACK=1
    - » Des options sont possibles
      - Taille maximale des paquets
      - Convention pour le contrôle de flux sur réseau haut débit ...
- **Ouverture**
  - **Protocole à trois phases:**



Automates de Mealy



Etablissement d'une connexion TCP

- Trois paquets sont nécessaires pour garantir qu'il n'y ait pas d'ambiguité sur des demandes de connexions et les données (doublons, timers de re-émission)
- Le numéro de séquence initiaux permettent de différencier des demandes de connexions dupliquées. Ils servent d'identificateur de connexion
- Flag Reset en cas de duplication (ou autre incohérence) : rejet
- Les numéros de séquence initiaux sont calculés à partir d'une horloge système de période 4 microsecondes. Cela garantit l'unicité d'un identificateur de connexion pendant 4 heures

Etablissement d'une connexion TCP

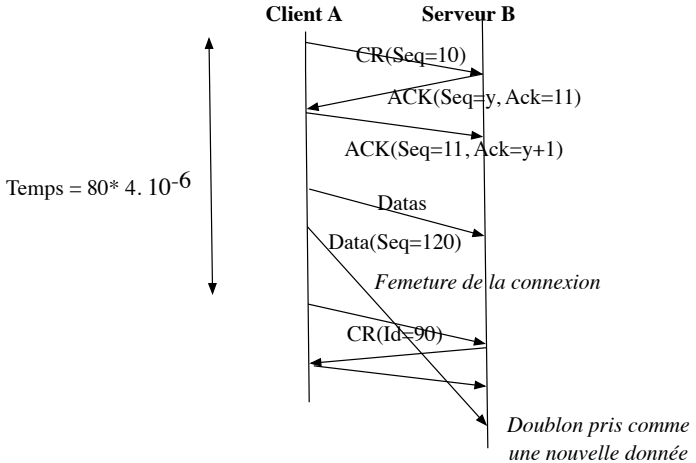
- Les numéros de séquences servent à numéroter aussi les octets de données
- **Problème: l'évolution des numéros de Séquence des données dépend du débit de la connexion**
  - On peut arriver à des cas où un doublon d'une ancienne connexion resurgisse avec un numéro de séquence suffisamment grand pour être pris comme une nouvelle donnée dans une autre connexion. On parle du problème de ***la zone interdite***)
- **Solution** : on attend la durée de vie maximale estimée sur Internet avant de ré-ouvrir une connexion sur la même paire de ports (~2 minutes)

Zone interdite

- Débit permettant d'arriver dans une zone interdite

  1 octet/4.10 $^{-6}$ sec= 8 bits/4*10^{-6}= 2 Mégabit/s

  On néglige le temps d'ouverture et de fermeture de la connexion

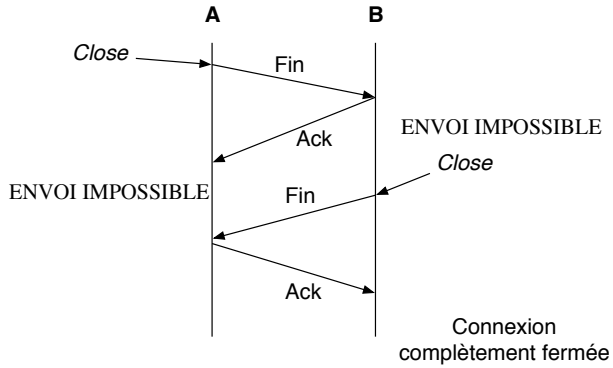


Fermeture de connexion

- **1ère solution:**
  - Fermeture brutale, quand on veut fermer la connexion d'un coté , on ne se soucie pas de l'état de l'autre. Comme au téléphone, on raccroche. Mais on peut s'être mis d'accord au niveau applicatif (''au revoir'' avant de raccrocher)
  - Primitive *close* des Sockets
- **2ème solution**
  - On veut pouvoir libérer les ressources d'émissions (tampons) d'un côté
  - Solution: la connexion est gérée comme deux demi-connexions unidirectionnelles
  - On peut fermer en émission, et continuer à recevoir, si l'autre n'a pas fini d'émettre
  - **Primitive *shutdown* des sockets**

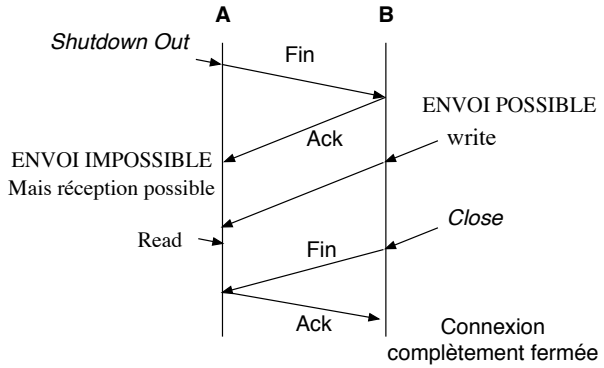
Fermeture de connexion
Le close

- Flag FIN pour signifier la demande de fermeture
- L'entité distante acquitte pour qu'il n'y ait pas d'ambiguité sur l'état de la connexion



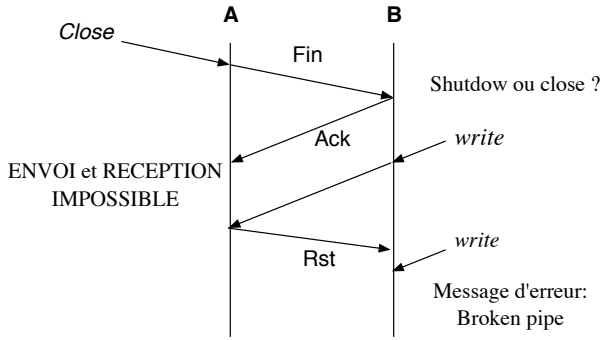
Fermeture de connexion
Le shutdown

- **Shutdown out, in, both**

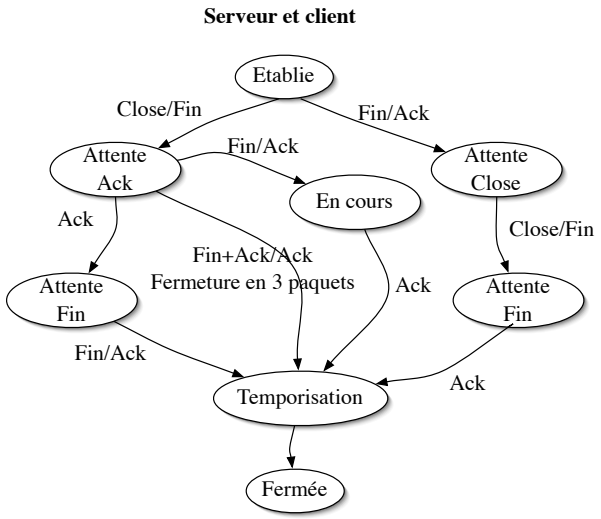


Fermeture de connexion
Un seul flag pour shutdown et close

- **Rien ne permet de différencier le shutdown et le close à la réception de Fin**



Automate de la fermeture



Principe de la récupération d'erreur dans TCP

» **Fenêtre d'anticipation avec re-émission sélective et acquittements "cumulatifs" (voir chapitre Contrôle d'erreur)**

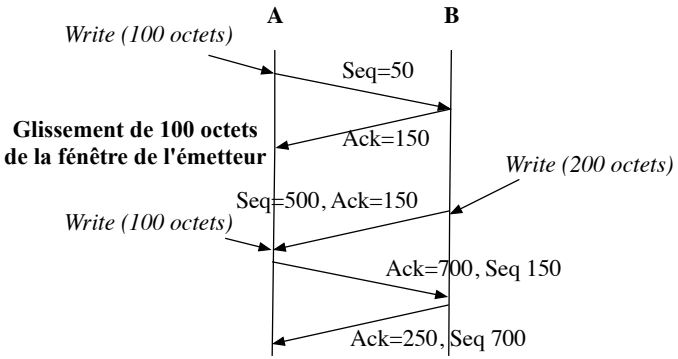
- Un timer par paquet, mémorisation côté émetteur des paquets jusqu'à qu'ils soient acquittés
- Un acquittement signifie "j'ai bien reçu jusqu'à ce numéro"
- Le récepteur mémorise les paquets non reçus en séquence
- Le timer de re-émission est calculé dynamiquement car la latence est très variable
- La taille du tampon nécessaire à la fenêtre d'anticipation de la récupération d'erreur est décidée par le programmeur côté émetteur
- La taille de ce tampon peut donc influer sur l'efficacité du protocole. Blocage fréquent en cas de tampon sous-dimensionné par rapport à la latence du réseau

Récupération d'erreur

- Flux d'octets (contrairement à UDP)
- Une fois la connexion ouverte, elle est symétrique et le transfert des données est bidirectionnel
- Numéro de séquence:
  - Numéro du 1er octet de donnée du message (sert aussi à la segmentation et réassemblage)
- Numéro d'acquittement:
  - Numéro de séquence + 1 du dernier octet bien arrivé (dans l'autre sens) (flag Ack=1)
  - Autrement dit prochain numéro de séquence attendu
- Un paquet peut servir à transporter des données et à acquitter un paquet du flux de sens inverse

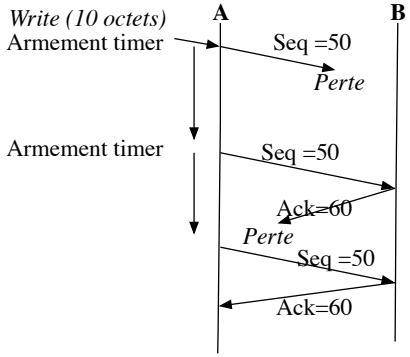
Exemple 1 de récupération d'erreur

- Le deuxième et le dernier paquet sont des acquittements "purs" mais il porte malgré tout un numéro de séquence
- Le quatrième paquet est un paquet de donnée qui sert aussi d'acquittement

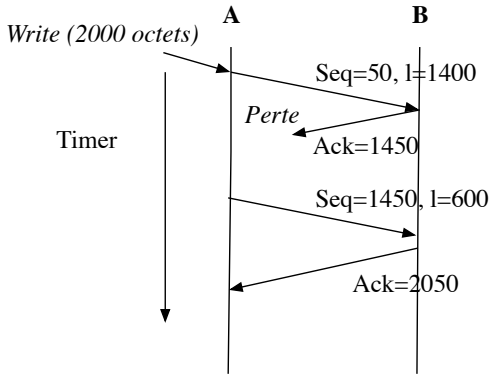


Exemple 2 de récupération d'erreur

- La perte d'un paquet de donnée déclenche la re-émission
- De même la perte d'un acquittement déclenche une re-émission



Exemple 3 de récupération d'erreur



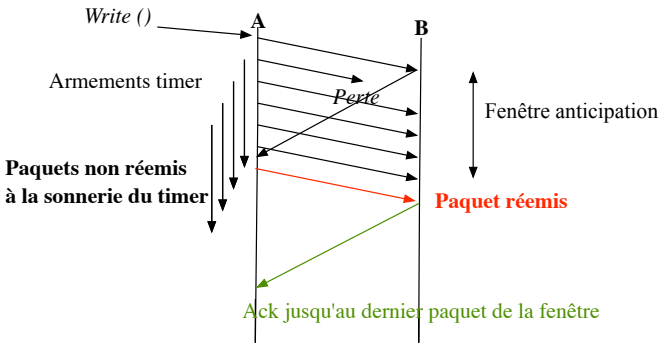
- Le deuxième acquittement acquitte les deux paquets de données précédent, il n'y a pas de re-émission à la sonnerie du timer (intérêt de l'acquittement "cumulatif")

Mécanisme de la récupération d'erreur

» **Côté récepteur:**
  » Si Numéro de séquence attendu alors acquittement jusqu'au dernier reçu
  » Sinon : mémorisation du paquet dans un tampon et acquittement jusqu'au dernier reçu sans « trou »
  » un Acquittement est envoyé à chaque réception d'un paquet (No du dernier reçu en séquence)

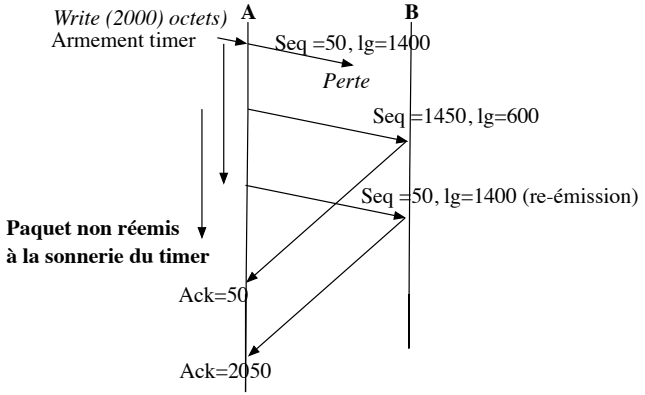
– **Côté émetteur:**
  » Timer associé à un paquet et mémorisation dans un tampon des paquets émis en attente d'acquittement
  » Si un timer sonne : re-émission du paquet de donnée associé
  » A la réception d'un acquittement : arrêt du timer
  » Attention: Un seul paquet en cours de re-émission (pour éviter les re-émissions inutiles)

Limitation des re-émissions



- Pour éviter les re-émissions inutiles dues à une perte dans la fenêtre à anticipation
- Un seul paquet est re-émis à l'expiration d'un timer
- Le dernier acquittement acquitte tous les paquets bien reçus en séquence

Exemple 4 de récupération d'erreur



- Le premier paquet est re-émis par expiration du timer
- Le deuxième ne sera pas re-émis tout de suite à l'expiration du timer, on attendra que le premier soit acquitté
- On évite ainsi des re-émissions inutiles (surtout quand la fenêtre est grande)

Calcul du timer de re-émission

» A régler au plus juste en fonction de la Latence du réseau

» Si timer trop court : re-émission inutile

» Si timer trop long : perte de temps en cas de perte

» Latence très variable : le timer est donc calculé dynamiquement en fonction du temps d'aller-retour des paquets de données et ACQ précédents

» Possible dès l'ouverture de la connexion

Fonctions de calcul du timer de re-émission

» A l'arrivée de chaque Acquittement donnant un RTTMesuré:

» Calcul d'une moyenne

  » **Erreur= Moyenne - RTTMesuré**

  » **Moyenne= Moyenne - A * Erreur**

  » A : "importance" des mesures les plus récentes

  » Dans la pratique (recommandé) A= 1/8

» Calcul d'une déviation (variation) des mesures

  » **Dev= Dev + B*(|erreur| - Dev)**

  » B recommandé à 1/4

» Calcul du timer: RTO (Retransmission Time Out)

  » **RTO= Moyenne + 4 * Dev**

  » Permet de ne pas sous-dimensionner le timer en cas de forte déviation

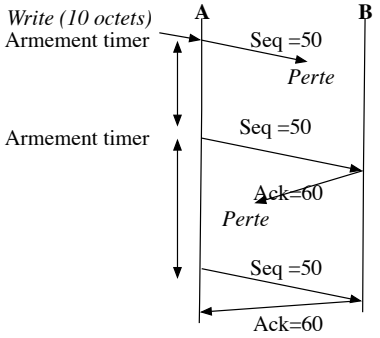
Timer de re-émission

» Souvent timer sur-estimé (finesse de l'horloge système, forte déviation). Optimisation dans TCP pour éviter d'attendre la sonnerie du timer (voir plus loin)

» En cas de re-émission la durée du timer est doublée

» En effet les pertes sont principalement dues à des saturations de routeurs et il est important de ne pas augmenter l'embouteillage par des re-émissions successives

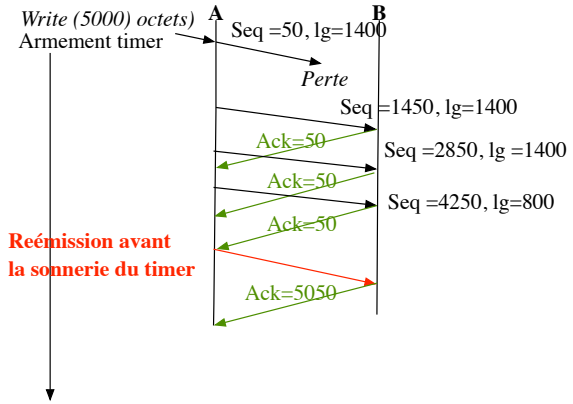
Exemple 6 de récupération d'erreur



- **En cas de re-émission la durée du timer est doublé pour ne pas augmenter l'embouteillage qui a causé la perte des paquets**

Retransmission rapide

» Intéressant car le timer est souvent surestimé
  » Si réception de 3 Acks dupliqués (portant le même numéro)
  » Cela indique une perte probable du paquet portant ce numéro de séquence
  » Alors re-émission immédiate du paquet de numéro de séquence égal au numéro d'Ack

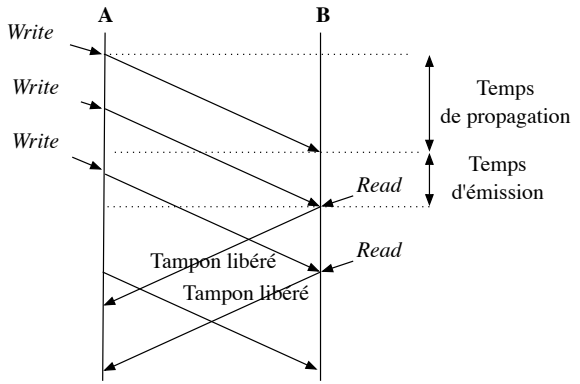


Contrôle de flux

- Problème: limiter le flux de l'émetteur pour ne pas saturer le récepteur
- Un tampon de taille fixe en réception: si le tampon est plein (l'application n'a pas encore récupéré les données) les prochaines données seront perdues
- Hypothèses:
  - La taille du tampon peut varier suivant les connexions (charge de la machine, nombre de connexions ouvertes...)
  - La latence est très variable au niveau transport, il est donc difficile de définir une taille de fenêtre optimale
  - La taille du tampon est fixée par le programmeur

Rappel : Contrôle de flux

- Acquittements permettant de signaler à l'émetteur la libération des buffers



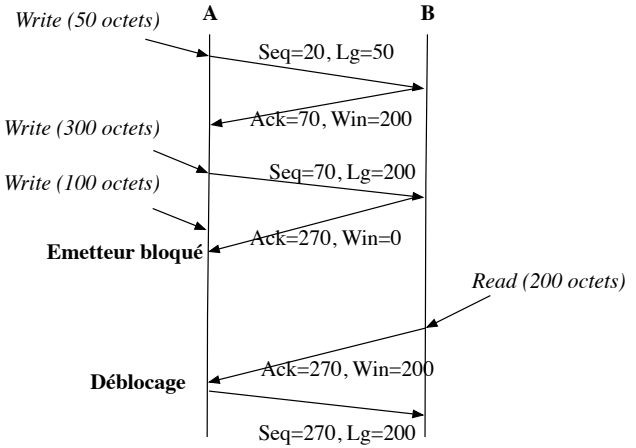
Mécanisme à nombre de crédits

- A la place de l'envoie d'acquittement spécifique au contrôle de flux TCP utilise un mécanisme à "***nombre de crédits***"
- TCP spécifie dans un acquittement de la récupération d'erreur le nombre d'octets libre dans le tampon de réception
- Permet à l'émetteur d'envoyer des paquets tant que le tampon du récepteur n'est pas plein
- Champ Fenêtre (WIN): nombre d'octets libre dans le tampon de réception et donc pouvant être expédiés après le numéro d'acquittement.

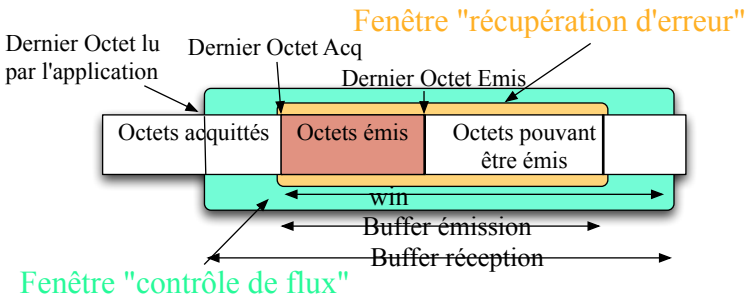
Mécanisme à nombre de crédits

- La taille des buffers est spécifiée dans le champ WIN des paquets de l'ouverture de connexion
- Quand WIN = 0 :
  - Le buffer du récepteur est plein
  - L'émetteur est bloqué jusqu'à la libération du tampon (lecture de l'application côté récepteur)

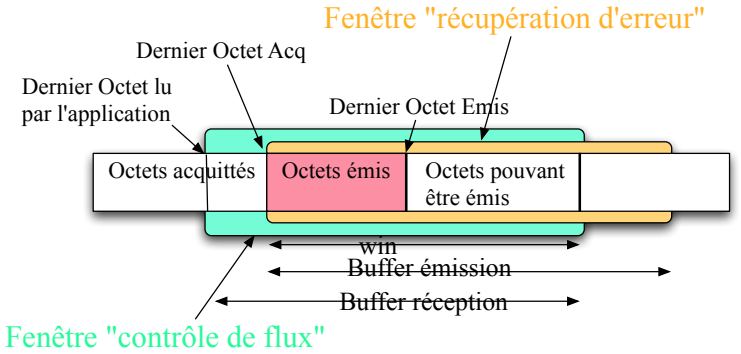
Exemple



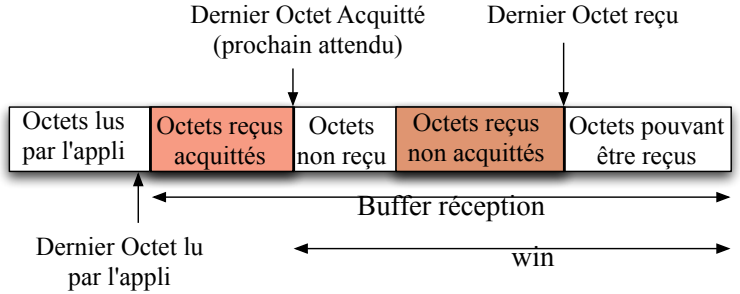
- A tout moment l'émetteur est bloqué en émission par la fenêtre la moins "avancée"
- Cas où la fenêtre du contrôle de flux est plus "avancée" que celle de la récupération d'erreur. L'émetteur va se "bloquer" sur la fenêtre de récupération d'erreur.



- Cas où la fenêtre du contrôle de flux est moins "avancée" que celle de la récupération d'erreur. L'émetteur va se "bloquer" sur la fenêtre du contrôle de flux



Côté récepteur

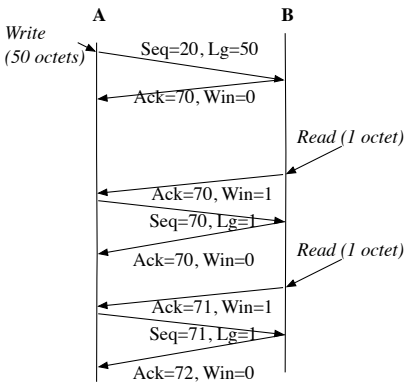


Contrôle de flux: choix de la taille du tampon de réception

- On peut essayer de régler la taille du tampon pour que les données soient émises au débit maximal si le récepteur suit le rythme (dépend du temps de propagation et du temps d'émission)
- Dans le cas d'un récepteur suivant le rythme, un tampon trop petit en réception impliquera des blocages fréquents en émission et donc des performances moindres
- Il ne sert à rien d'avoir la taille du tampon d'émission (contrôle d'erreur) supérieure à la taille du tampon de réception
- Il peut être intéressant d'avoir un tampon en réception plus grand qu'en émission si le rythme du récepteur est sporadique
- Sous free BSD: Taille Tampon réception= 2 * Taille Tampon d'émission

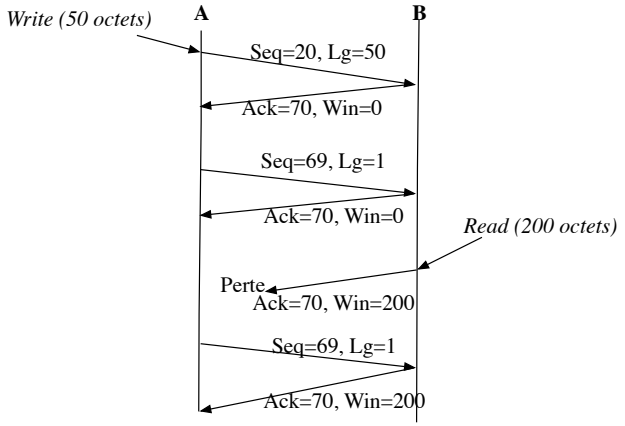
Cas de la fenêtre stupide

- Libération du buffer d'un seul octet
  - Déblocage par envoi d'un paquet avec champ Fen= 1
  - Envoi d 'un paquet de 1 seul octet de donnée
  - Charge du réseau inutile
- Solution:
  - Attendre une libération « importante » du buffer avant de débloquer
  - En pratique: Moitié du buffer ou taille maximale des segments



Cas de blocage

- Après un remplissage du tampon de réception si le paquet de "déblocage" se perd, on arrive à une situation de blocage
- Solution: l'émetteur continue à envoyer le dernier octet de donnée afin de forcer le récepteur à émettre un acquittement



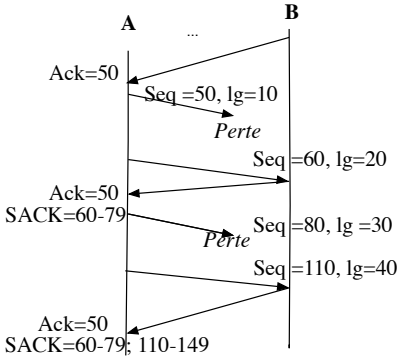
Options à l'établissement d'une connexion TCP

- Des notifications d'options sont possibles à l'ouverture de la connexion
- Elles apparaissent dans l'extension d'entête des 3 paquets d'ouverture de connexion
- Maximum segment size (MSS): permet de spécifier la taille maximale des paquets dans les deux réseaux de bout de connexion
- Time stamp: permet par estampillage temporel de connaître le temps aller-retour (round trip time: rtt) et d'initialiser le timer de reémission (voir récupération d'erreur)
- Valeur du décalage du champ WIN : permet d'augmenter la taille possible du champ WIN. Intéressant pour les réseaux haut-débit ou à temps de propagation important

Options à l'établissement d'une connexion TCP

- *Selective acknowledgment (sack)*: permet de spécifier des acquittements sélectifs
- Dans l'échange de données des acquittements peuvent porter dans l'extension d'entête des intervalles de numéro d'octets bien reçus après le numéro d'acquittement
- Exemple : NoACK= 12, (SACK= 50-60 ; 80-90)
- L'émetteur ne re-émettra que 12 à 49 et 61 à 79

Exemple d'utilisation des acquittements sélectifs



- Le récepteur sait que les octets de 60 à 79 et 110 à 149 ont été reçus
- Seul les octets de 10 à 49 et 80 à 109 sont re-émis
- Sans acquittements sélectifs TCP devrait attendre la sonnerie des timers des deux paquets perdus