Info0651 – Réseaux Informatiques

La couche Transport

Université de Reims Champagne Ardenne

Objectifs de ce cours

- Étudier les fonctions de la couche transport
- Comprendre le fonctionnement de deux protocoles : UDP et TCP
  - Fonctionnalités
- Discuter les défis technologiques avec l'arrivée des réseaux à très haut débit

Le service de Transport

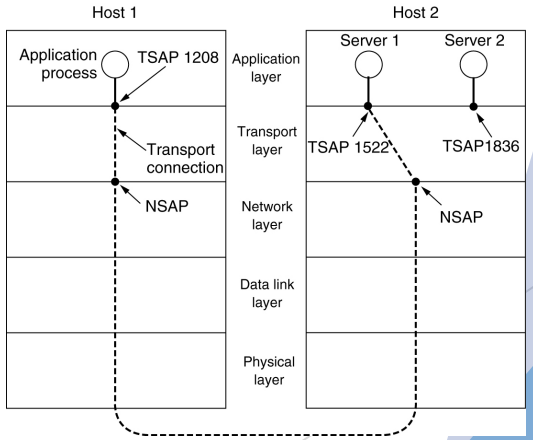
- Services destinés aux couches supérieures
  - Adressage
  - Établissement de connexions
  - Multiplexage
  - Contrôle de flux
  - Récupération de fautes
- Primitives de transport
  - Sockets

Primitives de base pour le transport

- Un service de transport simple doit au moins définir deux primitives
  - Send ()
  - Receive ()
- Suffisant pour transmettre des données qui rentrent dans un paquet IP
  - Certaines variations dépendent de la qualité de service exigé

Adressage à la couche 4

- L'adressage réseau (ex. IP) n'est pas suffisant pour assurer l'adressage bout-à-bout
  - Une adresse complémentaire est nécessaire pour retrouver l'application demandée
  - Ports :
    - Couche 4 TCP/IP
    - Couche 5 OSI

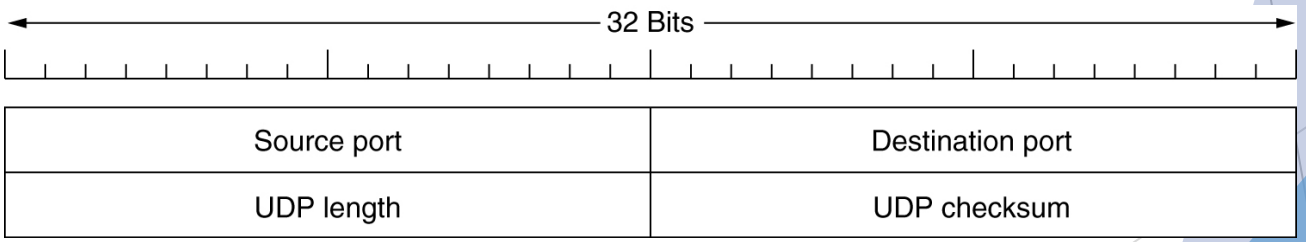


Protocoles Internet : UDP

- Introduction à UDP
- UDP et le contrôle de flux

Introduction à UDP

- L'en-tête UDP – uniquement l'essentiel
- Transport avec des garanties minimales
  - Aucun enchaînement entre les paquets (#seq)
  - Aucune garantie de livraison



UDP

- Utilise IP pour l'acheminement
  - Livraison non fiable

  + par rapport à IP : distinguer plusieurs applications destinataires
- Non connecté
  - Pas d'accusé de réception
- Pas de garantie de l'acheminement effectif
  - messages perdus, duplication, retard, livraison en désordre, ...
  - C'est à l'application de gérer tout ça

UDP et le Contrôle de Flux

- UDP = mode sans connexion
- Comment effectuer le contrôle de flux ?
  - Réponse : UDP ne le fait pas
  - Conséquences :
    - "remplissage des tuyaux" – best-effort
- Sans contrôle de flux → sans contrôle de congestion
  - Une transmission UDP consomme tous les ressources disponibles
    - Algorithme gourmand
  - Plus récemment : applications *UDP-friendly*

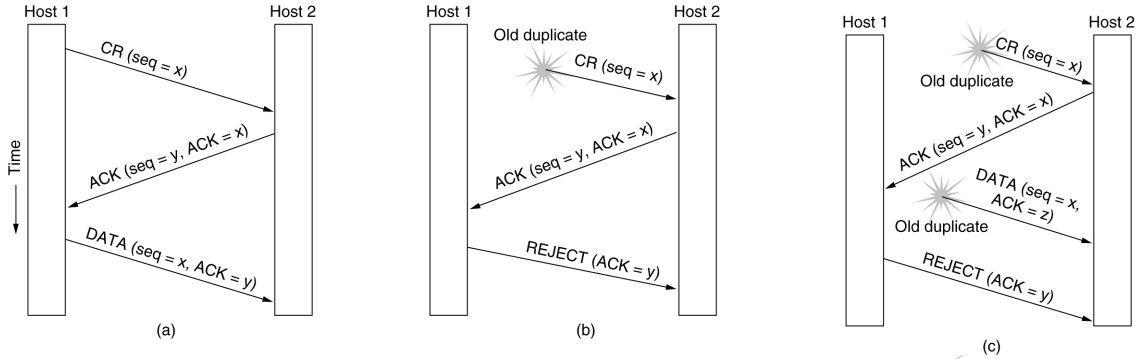
Transport en mode Avec Connexion

Pourquoi "Avec Connexion"

- Etablir une connexion a certains avantages
  - Garantie de "je suis là"
  - Identification de l'émetteur et du récepteur
  - Possibilité de contrôler un flux de données
    - Signaler des limites
    - signaler des pertes
- Inconvénients
  - On ne peut pas parler à un groupe
  - La gestion de la connexion a un coût

Établissement des Connexions

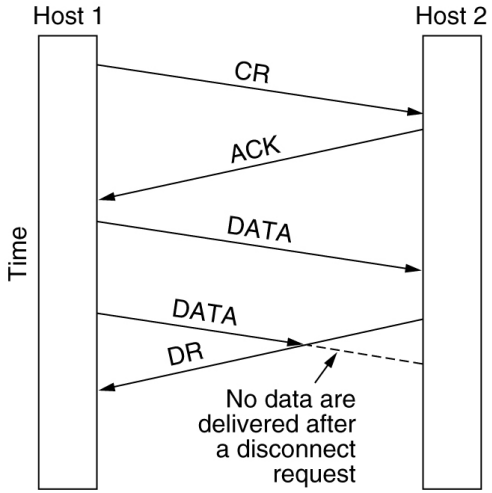
- Le premier défi est d'établir une connexion
  - Les deux côtés doivent être sûrs que la connexion est réussie
- Protocole de la "poignée de mains en trois étapes"
  - Efficace, mais il faut utiliser des ressources supplémentaires (#séq) pour éviter des surprises



Fermeture d'une Connexion

- Deux possibilités
  - Déconnexion asymétrique
    - La connexion est abruptement rompue
    - Risque de perte de données
  - Déconnexion symétrique
    - Les deux côtés se mettent d'accord sur la fermeture
    - Où ? Quand ? Et si un problème arrive ?

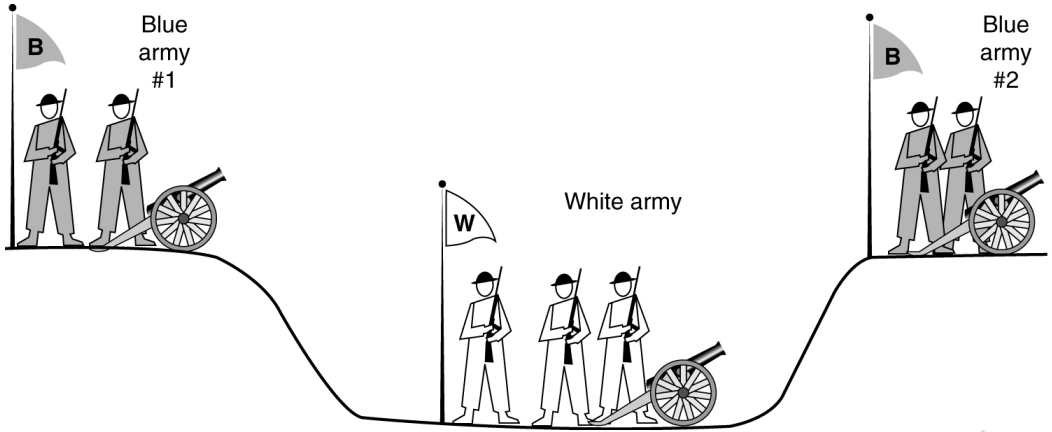
Fermeture d'une Connexion



Déconnexion abrupte

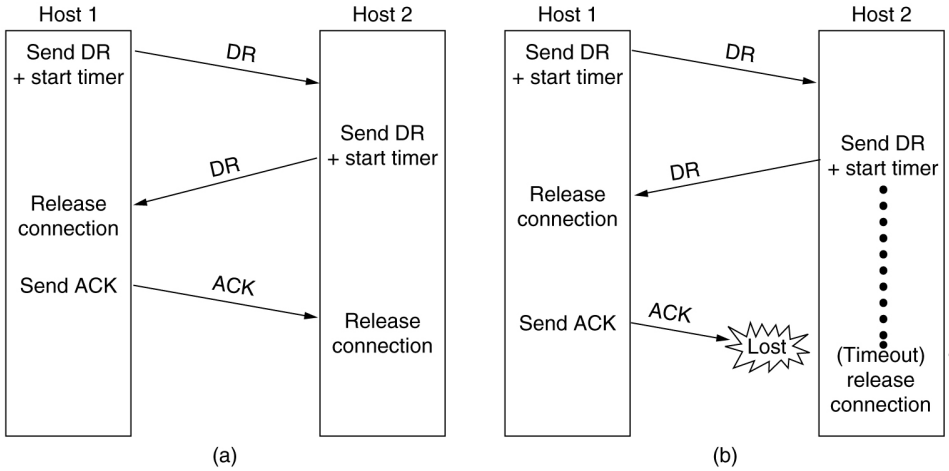
Fermeture d'une Connexion

- Le problème des deux armées
  - Simplification du problème des Généraux Byzantins
    - Pour réussir, il faut que les deux troupes de B attaquent au même temps



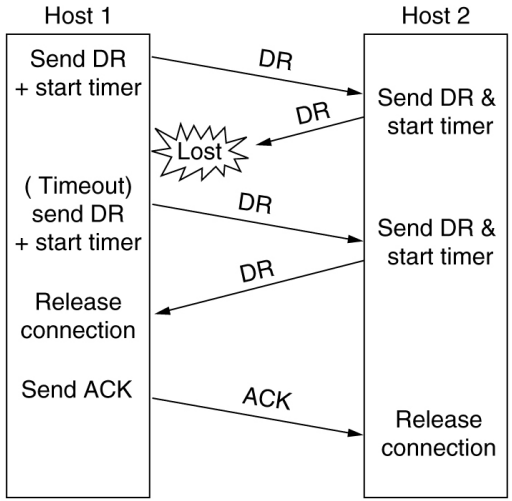
Fermeture d'une Connexion

- Quatre scénarios possibles
  - (a) Opération réussie
  - (b) Dernier ACK perdu

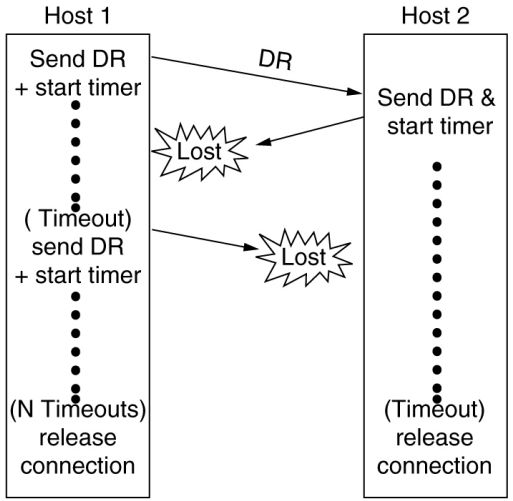


Fermeture d'une Connexion

- (c) Réponse perdue
- (d) Plusieurs échanges perdus - timeout



(c)



(d)

UNIVERSITÉ DE REIMS CHAMPAGNE ARDENNE

Contrôle de Flux et Buffering

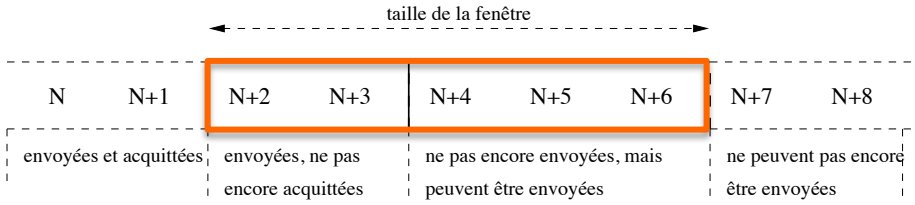
- Quantité de données qui varie selon l'application
- Différentes politiques de buffering et de contrôle (ex UDP vs TCP)
  - selon le type de service (avec ou sans connexion) il faut retransmettre les paquets manquants
  - si le service le demande, la couche 4 doit garantir que toutes les données sont arrivées (et dans le bon ordre)
- Fenêtres de taille variable (TCP)
  - Impact : l'allocation des buffers ne peut plus être faite de manière uniforme

Fenêtre d'Anticipation

- Principe
  - Permettre la **transmission à l'avance** d'un certain nombre de messages **avant acquittement**
  - **Limiter ce nombre** à un seuil
- Les fenêtres d’anticipation nécessitent des numéros de séquence pour les messages ainsi qu’une taille de fenêtre maximale pour l’émetteur et le récepteur

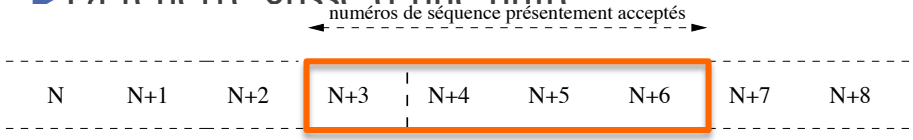
Fenêtre d'Envoi

- Contient des paquets qui ont été envoyés ou qui peuvent être envoyés
  - Ces paquets n'ont **pas encore été acquittées**
- Lorsqu'un paquet arrive de la couche supérieure
  a) Le prochain numéro de séquence est attribué
  b) L'entête de la fenêtre avance d'une unité
- Lorsqu'un acquittement arrive
  - La queue de la fenêtre avance d'une unité



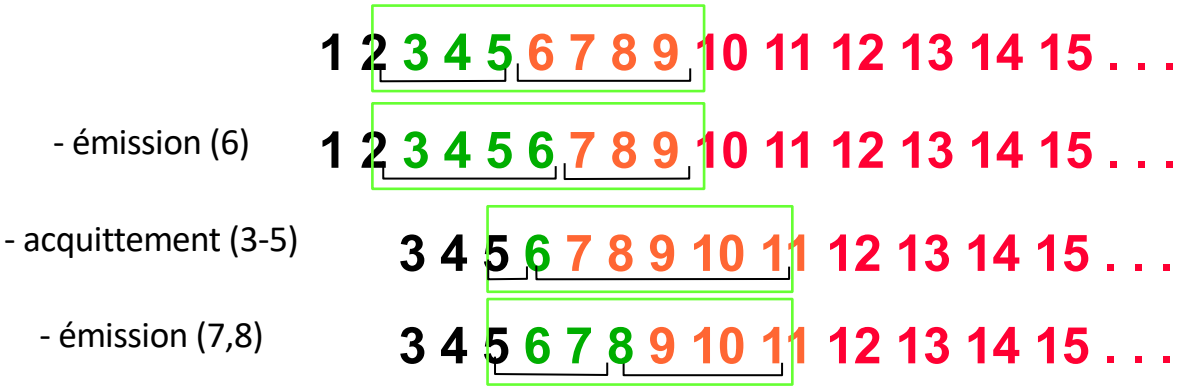
Fenêtre de Réception

- Contient les paquets qui peuvent être livrés
  - Les paquets ne sont livrés que dans l'ordre FIFO
  - Les paquets qui dépassent la fenêtre sont supprimées (*drop*)
- Lorsqu'un paquet contient un numéro qui correspond à la fenêtre
  - Selon la politique d'acquittement :
    - Le paquet est transmise à la couche supérieure
    - Un acquittement est crée
    - La fenêtre glisse d'une unité



Différentes Solutions pour le Contrôle de Flux

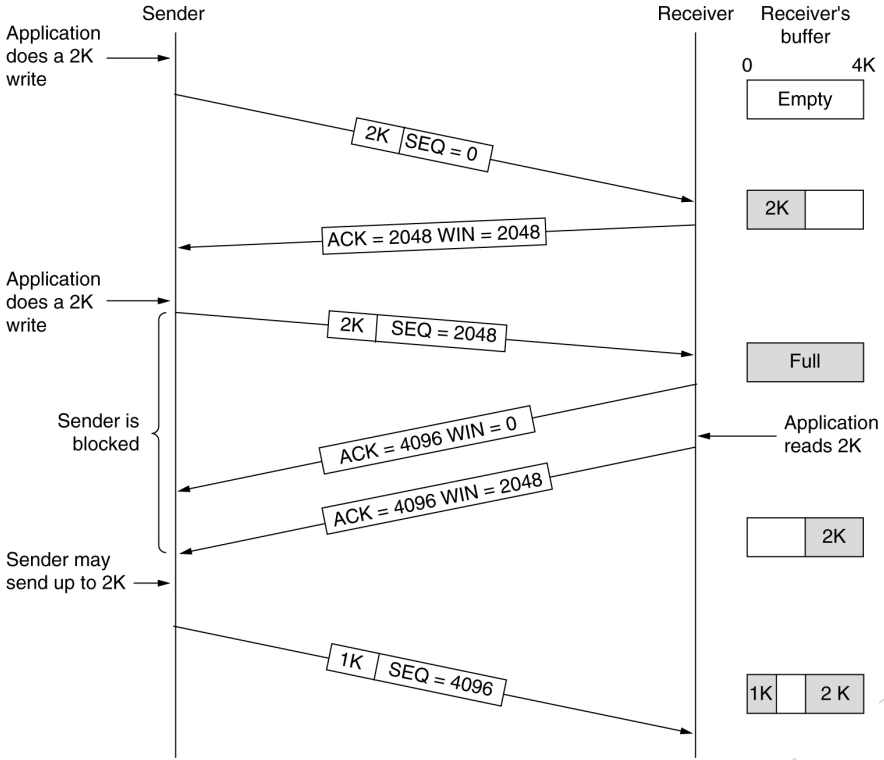
- Solution 1 : Utilisation de fenêtre glissante (taille fixe)
- le destinataire régule le flux en ralentissant les acquittements



Différentes Solutions pour le Contrôle de Flux

- Solution 2 : Utilisation de fenêtre glissante de taille variable
  - Mécanisme de crédit donné par le récepteur
- Certaines unités de données ont un champ supplémentaire qui contient **le crédit**
  - nombre d'unités de données que l'émetteur peut émettre en anticipation
- Permet de "commander" la taille des fenêtres des émetteurs

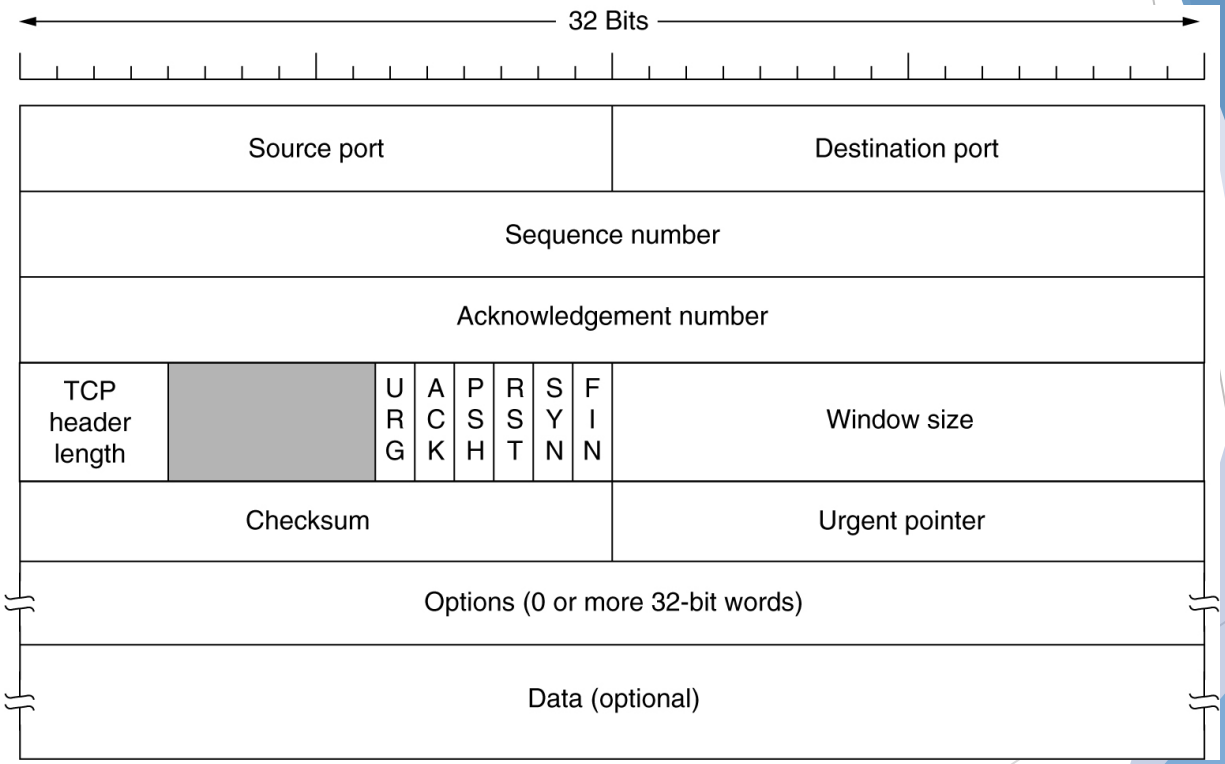
Le Mécanisme de Crédit (fenêtres glissantes variables)



Le protocole TCP

- Introduction à TCP
- Établissement de connexions TCP
- Fermeture de connexions TCP
- Politiques de retransmission

L'en-tête TCP

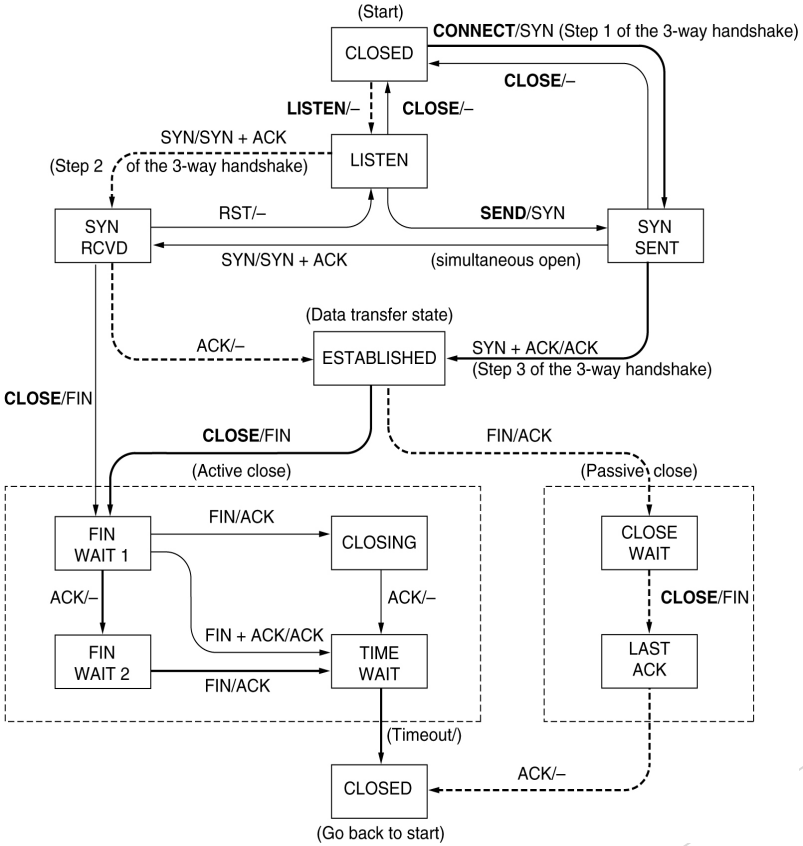


Gestion des Connexions

- Les états utilisés dans la machine d'états TCP

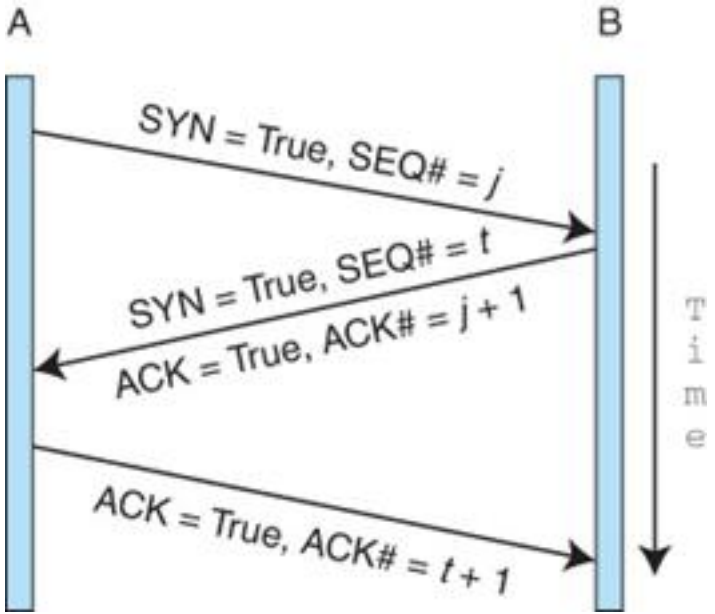
State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

Gestion des Connexions



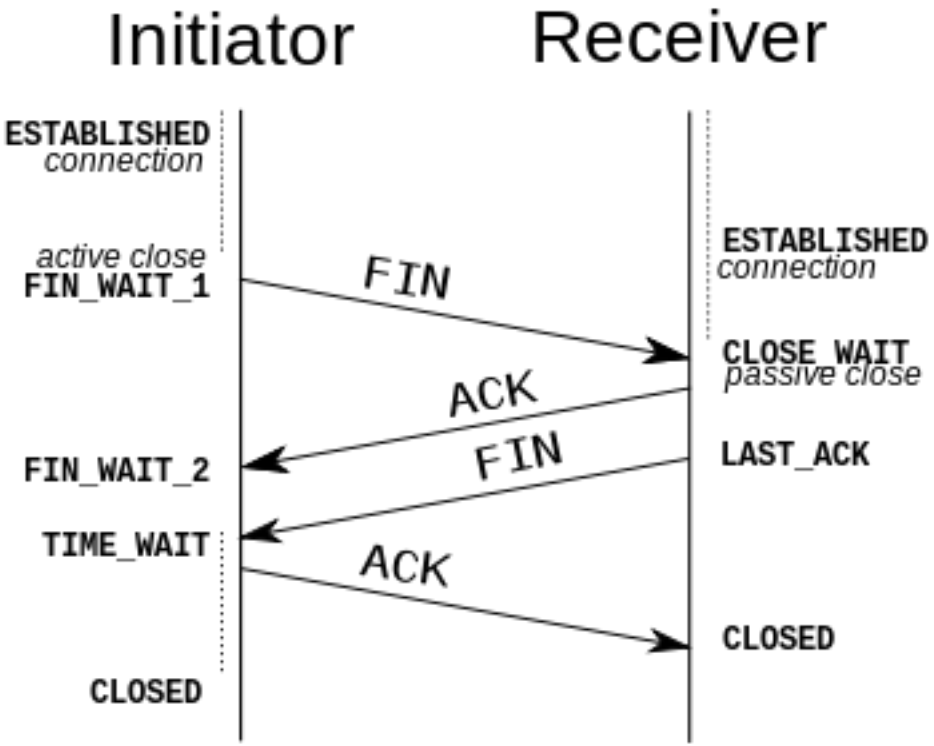
UNIVERSITÉ DE REIMS CHAMPAGNE ARDENNE

Ouverture de connexion



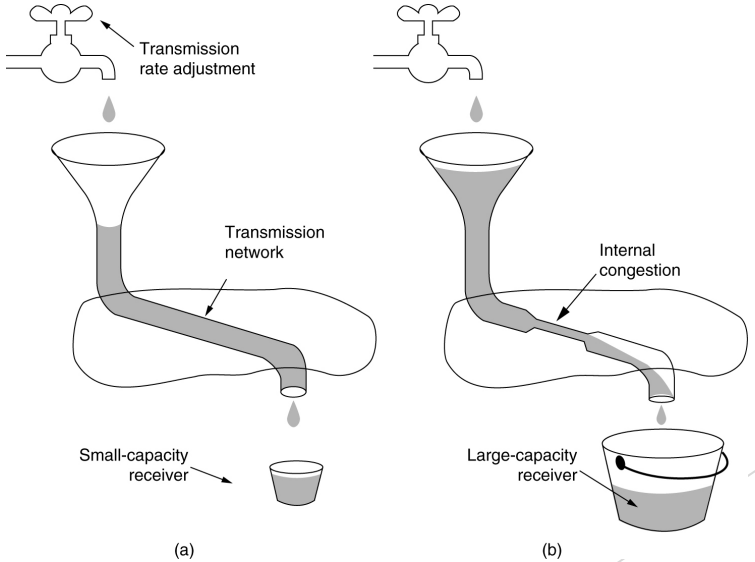
UNIVERSITÉ DE REIMS CHAMPAGNE ARDENNE

Ferneture de connexion



Contrôle de Flux en TCP

- Deux problèmes à gérer
  - Congestion à la réception
  - Congestion lors du transport



Contrôle de Flux

- Le mécanisme de des fenêtres glissantes avec crédit est efficace, mais il faut l'utiliser correctement pour éviter la congestion
- Questions : quelle est la capacité du réseau ?
  - a priori, les paires ne connaissent pas la vitesse du réseau
  - Solution : garder une deuxième fenêtre, la "**congestion window**" qui surveille la capacité du réseau
- Lors d'une transmission, l'émetteur doit respecter la fenêtre la plus petite

La Stratégie TCP - AIMD

- AIMD – Additive Increase, Multiplicative Decrease
  - La taille de la fenêtre dépend des paires mais aussi du niveau de congestion enregistré
- L'émetteur reçoit des indications implicites (perte de paquets) ou explicites (nouvelle taille de fenêtre) sur la congestion
  - définition d'une variable CongestionWindow (cwnd)

MaxWindow = min (**CongestionWindow** , AdvertisedWindow)

EffectiveWindow = MaxWindow – (LastByteSent -LastByteAcked)

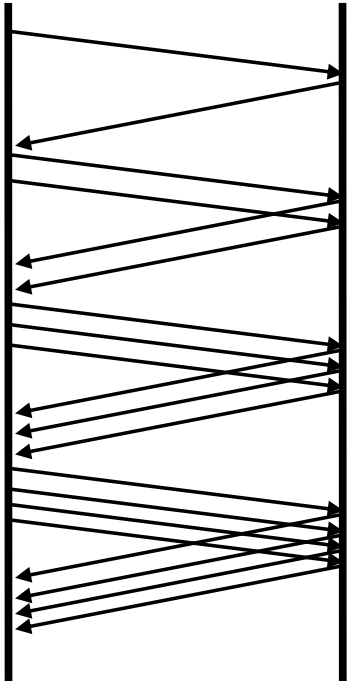
Additive Increase

- Idée de Base : à chaque paquet émis avec succès, augmenter la taille de la fenêtre de congestion (*cwnd*) d'une unité (crédit reçu dans un ACK, par exemple)
- En pratique, *cwnd* est augmenté proportionnellement à la taille maximale du segment (MSS) à chaque ACK arrivé

increment = MSS x (MSS / cwnd)

cwnd = cwnd + increment

Additive Increase



Multiplicative Decrease

- Le principe : un message perdu (timeout) est probablement dû à la congestion du réseau
  - Pour éviter des nouvelles pertes, TCP doit diminuer la fenêtre d'envoi
- Multiplicate Decrease : à l'expiration d'un timeout TCP divise en deux la valeur de la variable *cwnd*
  - La fenêtre peut être divisée successivement tant qu'il y a de la congestion
  - cwnd ne peut pas être plus petite que la taille d'un paquet

Comportement typique AIMD

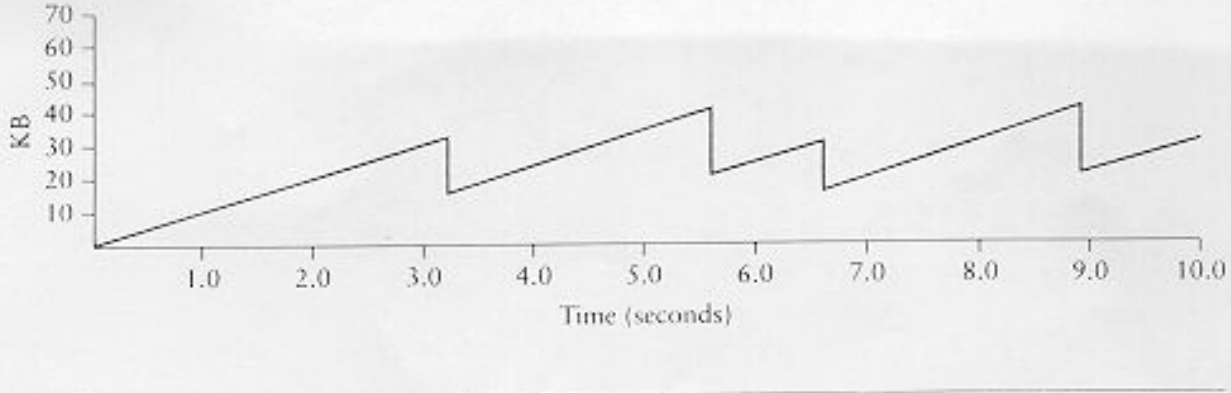


Figure 6.9 Typical TCP sawtooth pattern.

AIMD

- AIMD est un contrôle de congestion nécessaire pour que TCP puisse fonctionner de manière stable
  - Il est important d'avoir des mécanismes précis pour définir les timeouts, afin d'éviter le déclenchement sauvage de *multiplicative decrease*
- Les timeouts sont généralement établis en fonction du temps d'aller-retour moyen (RTT)

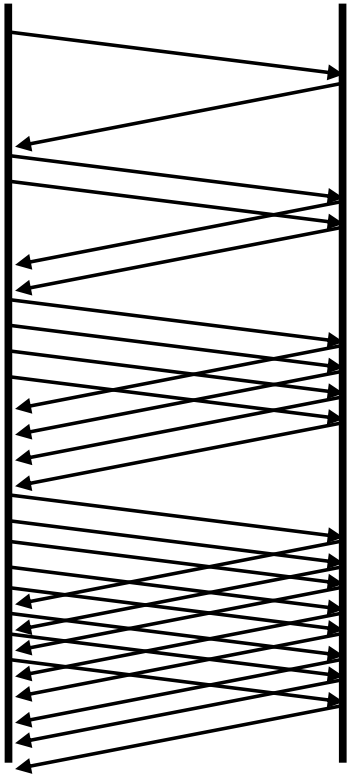
Slow Start

- Le mécanisme linéaire de l'*additive increase* est trop lent lors du démarrage à partir du zéro (*cold start*)
  - À partir de la version TCP nommée "Tahoe", un nouveau mécanisme a été introduit
- **Slow start** : mécanisme qui permet une augmentation exponentielle de la taille de *cwnd* lors du démarrage
  - ATTENTION : "slow start" sert à prévenir le démarrage lent
  - Ce mécanisme est toujours plus lent que l'envoi d'une fenêtre entière dès le début (*advertised window*)

Slow Start

- La source commence la transmission avec *cwnd*=1
- À chaque ACK reçu, *cwnd* est augmenté
  - en pratique, *cwnd* est doublé à chaque période RTT
- Ce mécanisme est déclenché dans deux situations :
  - au début d'une connexion (*cold start*)
  - lorsque la connexion est bloquée en attendant un timeout (la valeur de *l'advertised window* devient zéro !)
    - dans ce cas TCP peut déduire une "borne de congestion", afin d'arrêter le *slow start* à temps

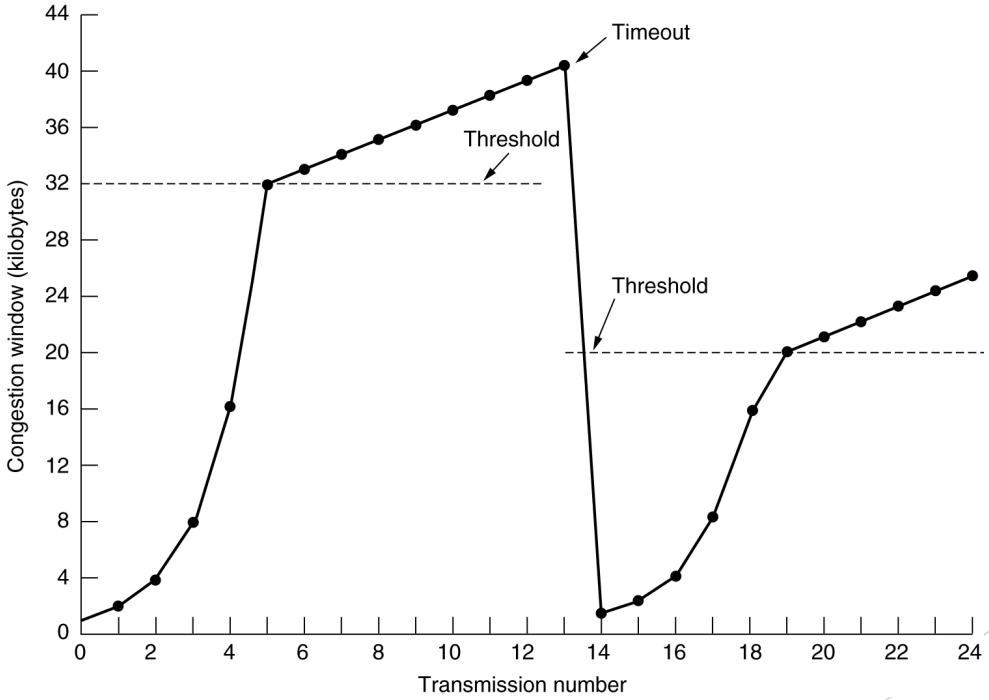
Slow Start



Congestion Avoidance

- Lorsqu'on connaît la "borne de congestion", TCP arrête *Slow Start* et entre dans une phase de augmentation progressive connue sous le nom de ***Congestion Avoidance***
  - Éventuellement un paquet sera perdu par hasard ou parce qu'on l'a émis trop vite
- Quand cela arrive, nous réduisons la valeur de la "borne de congestion"
  - on relance le mécanisme *Slow Start*

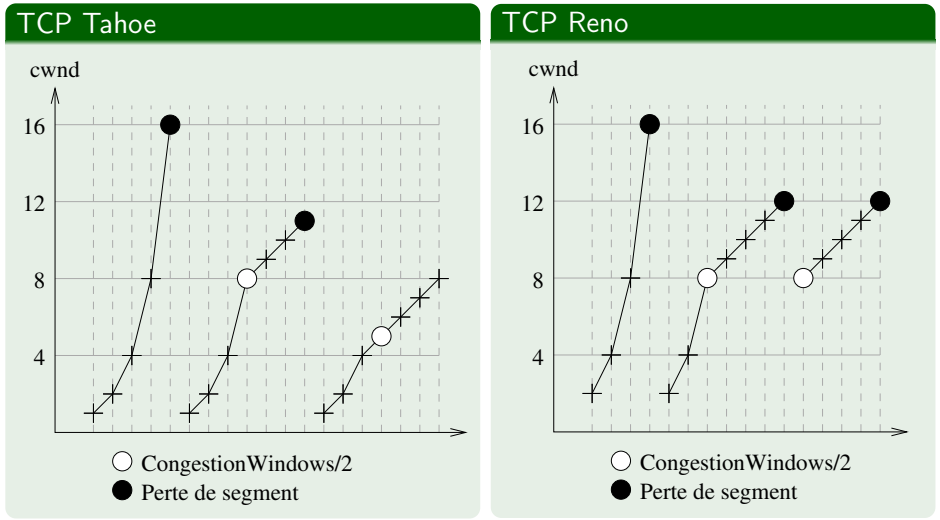
Contrôle de Flux TCP Tahoe



Évolutions de TCP

- Des nouvelles versions de TCP tentent d'améliorer les mécanismes AIMD
  - Versions toujours compatibles
- Ex : TCP Reno
  - La taille de départ de la fenêtre est à 2
  - Comportement différent suivant le mode de détection de perte :
    - Perte détectée suivant les timeouts
      - Slow Start
    - 3 réceptions d'ACK dupliqués
      - Congestion Avoidance

TCP Tahoe vs Reno

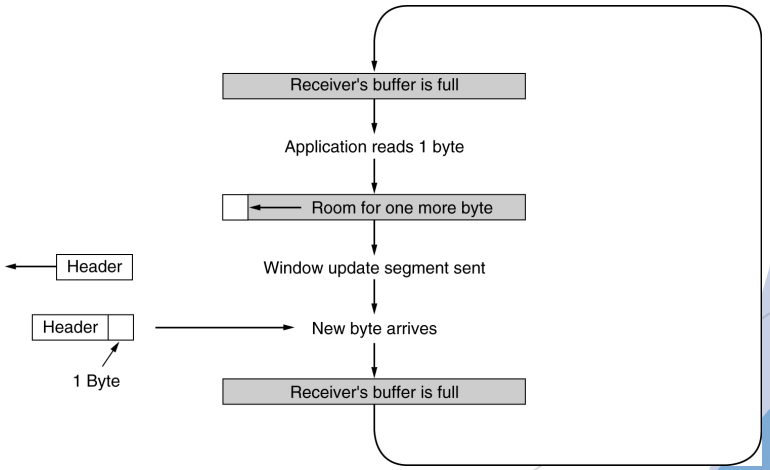


Techniques Supplémentaires

- Algorithme de Nagle
  - Regroupement de "petits paquets" pour réduire la charge de mise sur le réseau
    - Envoi d'un premier octet dès son arrivée et stockage des prochains octets tant que le premier paquet n'est pas acquitté
    - Technique à désactiver pour certaines applications
- Algorithme de Clark
  - Empêcher le syndrome de la "fenêtre stupide"

Syndrome de la fênetre stupide

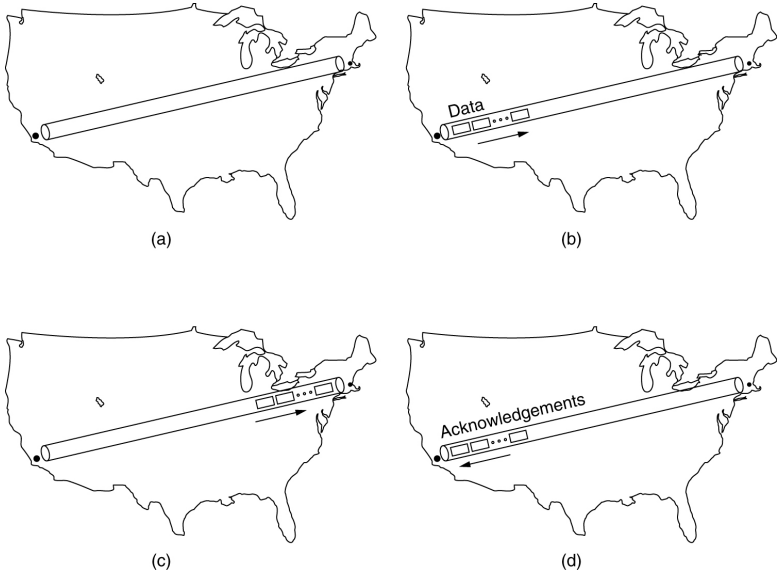
- *Silly window syndrom*
  - Si on libère la fenêtre par des petits blocs, on ne fait que plomber la performance
  - Il faut attendre que la fenêtres se libère d'une quantité suffisante



Problèmes typiques de performance

- Congestion
- Déséquilibre de ressources
- Surcharge temporaire (rafales)
  - Ex : si les destinataires multicast répondent à un paquet erroné
  - Ex : redémarrage des machines après une coupure de courant
- Temporisateurs/Buffers mal dimensionnés
- Écart type des paquets

Le problème des temporisateurs

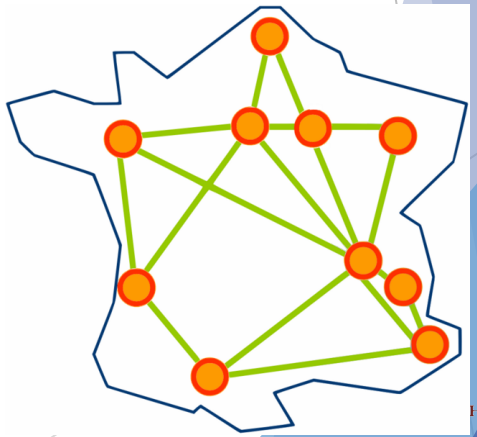


La transmission d'un 1Mo de San Diego à Boston

(a) t = 0, (b) après 500 µsec, (c) après 20 msec, (d) après 40 msec.

Adéquation des Fenêtres

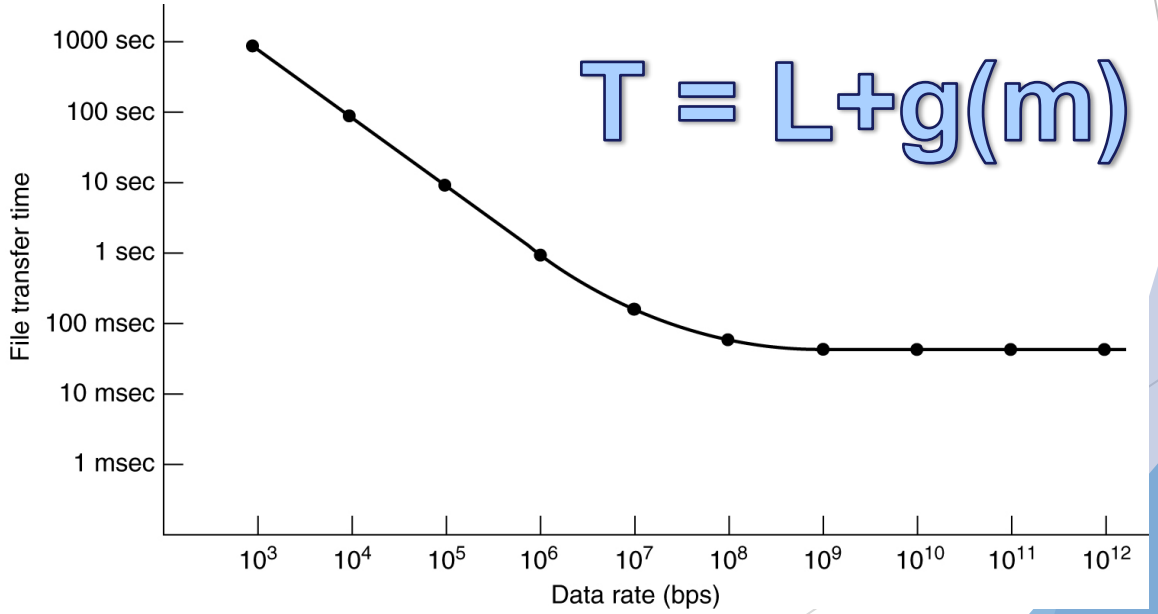
- La taille des fenêtres doit être au moins équivalente à
  - BDP = RTT * (débit théorique / 8)
- Attention : une fenêtre trop grande retarde la détection des pertes
  - Exemple : Grid'5000
    - RTT = 22ms entre Sophia et Rennes
    - Lien à 1Gbit/s
      - buffer=131072 bytes 69.05 Mbits/s
      - buffer=524288 bytes: 255.79 Mbits/s
      - buffer=2097152 bytes: 835.87 Mbits/s
      - buffer=3145728 bytes: 870.07 Mbits/s
      - buffer=4194304 bytes: 897.66 Mbits/s



UNIVERSITÉ DE REIMS CHAMPAGNE ARDENNE

Protocoles pour les réseaux Gigabit

Temps de transfert d'un fichier de 1Mo sur une ligne de 4000 km (40ms de latence)



Les défis des réseaux Gigabit

- TCP/IP est un protocole qui vieillit
  - Spécification datant de 1980 !
- Un débit accru met en péril certaines "constantes" TCP
  - Ex : les numéros de séquence TCP = 2^{32}
    - À 56 kbit/s → plus d'une semaine
    - À 10Mbit/s → 57 minutes
    - À 1Gbit/s → 34 secondes
    - Mais les temporisateurs de déconnexion restent à 120s !
  - Des solutions sont proposées dans la RFC 1323

En résumé

- Comprendre la différence entre les services liaison de données, réseau et transport
- Comprendre les services offerts par TCP et UDP
- Comprendre les limitations et défis pour les années à venir