

Réseaux Couche Transport

E. Jeandel

Emmanuel.Jeandel at lif.univ-mrs.fr

- Couche réseau :
 - Transmission précise de données entre deux machines
 - Non fiable

Couche transport

- Transmission de données entre deux *processus*
- Fiable ou non

1 Multiplexage

2 TCP

- La couche réseau permet la transmission entre deux machines
- Deux processus différents sur la machine *A* veulent parler à un deux processus sur la machine *B*.
- La couche réseau ne permet pas de les distinguer

- *Ports*
- ~ Adresses de transport
- Sur 16 bits.
- Communication identifiée par un quadruplet
(Adresse Réseau A, Port A, Adresse Réseau B, Port B)
- Triplet du point de vue de chacun des participants

- Si la machine A veut parler à la machine B sur le port b , elle doit d'abord choisir un port a
- Comment NAT peut-il fonctionner ?
- PAT (Port Address Translation)

- Si la machine A veut parler à la machine B sur le port b , elle doit d'abord choisir un port a
- Comment NAT peut-il fonctionner ?
- PAT (Port Address Translation)

- Ajoute les ports à IP, et c'est tout
- Encapsulé dans IP

16	16	16	16	
Port Source	Port Destination	Longueur	CRC	Données

Checksum : parité sur 16 bits de UDP+un header IP

1 Multiplexage

2 TCP

- UDP ne suffit pas
- Aucune garantie que les paquets arrivent, ni qu'ils arrivent dans l'ordre
- Besoin de transfert fiable

- Protocole connecté
- A la création, on alloue des structures de chaque côté pour s'occuper spécifiquement de la connexion
- Fiable
- Contrôle de la congestion

L'unité de base en TCP s'appelle le *segment* MSS : Max Segment Size

1 Multiplexage

2 TCP

- **Fiabilité**
- Contrôle de flux
- Qualité de service
- Connexion
 - **Création**

- TCP utilise une variante de GoBackN
- Contrôle au niveau des *bits* et non pas des segments
- ACK n : j'ai tout reçu jusqu'au n -ième bit non compris
- Pas de NACK

- Premier numéro choisi aléatoirement et envoyé au correspondant lors de l'établissement de la connexion
 - “empêche” de hacker une connexion TCP
- Numéros sur 32 bits

Optimisations

- Optimisation : *fast retransmit* de n si 3 ACK n .
- Piggybacking (Delayed ACK)
 - Attendre 200ms avant de réenvoyer un ACK, au cas où il y aurait une réponse sur laquelle on peut se greffer
 - Si un deuxième paquet arrive, envoyer le ACK
 - En régime “permanent”, seulement un paquet sur deux sera acquitté

Timeout

- Comment estimer le timeout avant de réenvoyer ?
- Estimation du temps AR (ERTT : estimated round-trip time)
-

$$ERTT := \alpha ERTT + (1 - \alpha)RTT$$

- RTT : temps AR du dernier paquet acquitté
- $0.8 \leq \alpha \leq 0.9$
- Comment fixer le timer ?

$$TIMER := ERTT + 4Deviation$$

$$Deviation := \alpha Deviation + (1 - \alpha)|RTT - ERTT|$$

1 Multiplexage

2 TCP

- Fiabilité
- Contrôle de flux
- Qualité de service
- Connexion
 - Création

- Chacun des deux participants a un buffer de réception
- Prévenir que le buffer est presque plein
- B prévient A qu'il lui reste p bits disponibles dans son buffer à chaque message qu'il envoie
- A doit s'arranger pour que

$$\text{DernierBitEnvoye} - \text{DernierBitAcquitte} \leq p$$

Problème

v1

- B envoie un message comme quoi $p = 0$
- Que se passe-t-il ?
- Solution : A envoie des paquets de 1 bit que B acquitte.
- *Persistence Timer*

Problème

v1

- B envoie un message comme quoi $p = 0$
- Que se passe-t-il ?
- Solution : A envoie des paquets de 1 bit que B acquitte.
- *Persistence* Timer

Problème

v2

- B est beaucoup plus lent que A
- En régime “permanent”, chaque segment fera un seul octet
- *Silly Window Syndrome*

Solution au SWS

Côté destinataire

- Attendre que p redevienne suffisamment grand
- Par exemple taille max d'un segment ou la moitié de la taille totale
- Que faire en attendant ?
 - Ne pas envoyer de ACK
 - Envoyer des ACK avec $p = 0$

Problème côté émetteur

v3 tinygram

Eviter les petits paquets niveau émetteur (*Algorithme de Nagle*)

- Si beaucoup de données (plus que MSS), les envoyer
- S'il y a des données non acquittées, attendre
- Sinon envoyer

Attention

- Peut être indésirable (mouvement de la souris)
- Conflit avec le Delayed ACK
- Peut être désactivé (`TCP_NODELAY`)

Exemple

- La taille maximum d'un segment pour une pile réseau OSX est de 1448 octets.
- A, sous OSX, envoie 100000 octets à B puis attend le OK de B
- B attend les 100000 octets et envoie un OK à A.

Questions :

- Sous OSX, le programme va considérablement plus vite pour envoyer 102000 octets que 100000. Pourquoi ?
- Sous Windows qui a un segment max de 1460 octets, le programme va plus vite que sous OSX pour 100000 octets, mais moins vite pour 102000...

1 Multiplexage

2 TCP

- Fiabilité
- Contrôle de flux
- Qualité de service
- Connexion
 - Création

- Contrôler la congestion
- Essentiellement au niveau des routeurs qu'on ne contrôle pas
- Quelques infos par ICMP (globales !), mais aussi dans TCP lui même (par connexion)

Exemple

- A et B reliés par un routeur au monde extérieur
- Routeur parfait (buffers infinis, temps de traitement nul)
- Câbles de débit D
- Comportement du système en fonction du débit moyen d de A et de B ?

Exemple

- Débit sortant : $\min(d, D/2)$ par connexion
- Délai (temps moyen avant réception) : $\sim 1/(D - 2d)$
- Dans la réalité, gros délai \rightarrow réémission des paquets..

Exemple

v2

- Même exemple
- Réémission des paquets → délais encore plus grands
- Nombre de paquets transférés de plus en plus petit
- Le routeur surchargé l'est encore plus. . .

Dès que le débit est trop élevé

- Délais importants
- Retransmission de segments dans la file d'attente d'un routeur
- Paquets dupliqués envoyés par le routeur
- Les paquets abandonnées par le routeur correspondent à du trafic gâché

- Qui doit s'en occuper ?
 - Couche réseau (tous les participants, routeurs compris)
 - Couche transport (uniquement les extrêmités)
- Choix TCP/IP : Dans la couche transport

Comment s'en occuper

- Prévenir plutôt que guérir
- Comment contrôler le débit dans TCP ?
 - En changeant la taille de la fenêtre...

Comment s'en occuper

- Prévenir plutôt que guérir
- Comment contrôler le débit dans TCP ?
 - En changeant la taille de la fenêtre...

Congestion dans TCP : AIMD

Additive Increase - Multiplicative Decrease

Point de vue : les pertes de paquets viennent uniquement des files d'attente dans les routeurs

- Fenêtre de congestion de taille F
- Seuil λ de congestion.
- Si $F < \lambda$
 - Au début, taille de la fenêtre de 2 MSS (*slow start*)
 - A chaque ACK, on augmente de 1 MSS
 - Augmentation *exponentielle*
- Si $F \geq \lambda$
 - Augmentation de 1MSS tous les F ACK
 - Augmentation *linéaire*
- Si perte de paquet (timeout) :
 - $\lambda = F/2$
 - La fenêtre passe à 1 MSS

Conséquence pour l'utilisateur

- A cause du *slow start*, il vaut mieux une connexion pour transférer deux fichiers que deux connexions consécutives.
- Sans oublier les coûts de connexion. . .
- Exemple : HTTP
 - Keep-Alive
 - Pipelining

- n machines derrière un routeur
- Stratégie du routeur : supprimer les nouveaux paquets quand la file est pleine
- Problème : quand la file d'attente est pleine, *toutes* les connexions vont être perturbées et redémarrer

RED

Random Early Detection/Drop/Discard

- File d'attente du routeur séparée en deux “moitiés”
- On remplit la première partie
- Si la première partie est pleine, on accepte dans la file d'attente un nouveau paquet IP qu'avec probabilité p
- Bien choisir p

1 Multiplexage

2 TCP

- Fiabilité
- Contrôle de flux
- Qualité de service
- Connexion
 - Création

- Débuter et terminer la connexion
- Difficulté en général, dans un modèle asynchrone avec pertes, de se synchroniser
- Problème des généraux byzantins

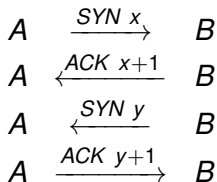
Création de la connexion

- Vérifier que les deux parties veulent communiquer
- Allouer les buffers et autres structures
- Se mettre d'accord sur les numéros de séquence

Création de la connexion

- Paquet *SYN* x : je commence à numéroté à x

Méthode informelle



Création de la connexion

- Paquet *SYN* x : je commence à numéroté à x

Méthode informelle

