Réseaux 6
La couche transport

Juliusz Chroboczek

18 octobre 2020

La couche transport

Application	(7)
Transport	(4)
Internet ou Réseau	(3)
Lien	(2)
Physique	(1)

La couche transport

Couche de bout en bout :

- implémentée dans les extrémités ;
- ne sait pas qu'il y a des liens.

Implémente des fonctionnalités généralement utiles pour les applications :

- multiplexage interne (obligatoire) ;
- segmentation (optionnel) ;
- communication ordonnée (optionnel) ;
- fiabilité (optionnel) ;
- contrôle de flot (obligatoire ?) ;
- contrôle de congestion (obligatoire ?).

Multiplexage interne

Un paquet IP est destiné à une interface qui appartient à un hôte.

Une application veut envoyer des données à une application tournant sur un hôte. Il faut donc distinguer celles-ci.

Numéro de port : entier de 16 bits. À la couche transport, données destinées à une paire (IP, p).

Adresse de socket : paire (IP, p). Une adresse de socket fait donc 48 ou 144 bits.

(Socket : structure de données, représente l'extrémité de la communication.)

Protocoles de couche transport

UDP : paquets, non-fiable, non-ordonné, non-contrôlé
comme IP.

TCP : flots d'octets, fiable, ordonné, contrôlé
le contraire d'IP.

UDP-Lite, DCCP, SCTP, QUIC etc.

Protocole UDP

UDP est une couche fine au-dessus d'IP :

- datagrammes (paquets) ;
- non-fiable ;
- non-ordonné ;
- pas de contrôle.

UDP implémente donc le même service que IP avec juste le multiplexage en plus.

UDP : format des paquets


```
 0       7 8      15 16     23 24     31
+--------+--------+--------+--------+
|     Source      |   Destination   |
|      Port       |      Port       |
+--------+--------+--------+--------+
|                 |                 |
|     Length      |    Checksum     |
+--------+--------+--------+--------+
```

UDP : applications

- Protocoles requête-réponse sans état (DNS, Kademlia) ;
- protocoles temps réel (NTP, jeux en ligne) ;
- protocoles multimédia (RTP, Skype) ;
- VPN (contrôle et fiabilité : protocole encapsulé).

UDP ≈ IP : on peut donc construire un protocole de couche transport au-dessus de UDP :

- QUIC (HTTP/3) ;
- µTP (BitTorrent) ;
- RTP (SIP, WebRTC).

Traverse les NAT, pas de modification du noyau.

Protocole TCP

TCP : couche épaisse :

- sémantique par flots d'octets ;
- fiable ;
- ordonné ;
- contrôle de flot et de congestion.

Tout le contraire d'IP !

Sémantique par flots d'octets

TCP : sémantique par flots d'octets :

- ce qui est préservé, c'est la suite d'octets transmis ;
- frontières des `write` non-préservées

Agrégation :

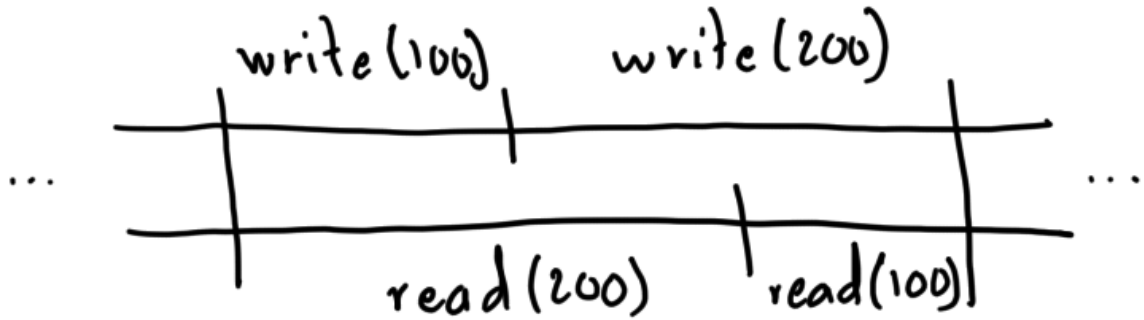
- `write(100), write(200)` ⟶ `read(300)` ;

Scission :

- `write(300)` ⟶ `read(200), read(200)`.

Sémantique par flots (2)

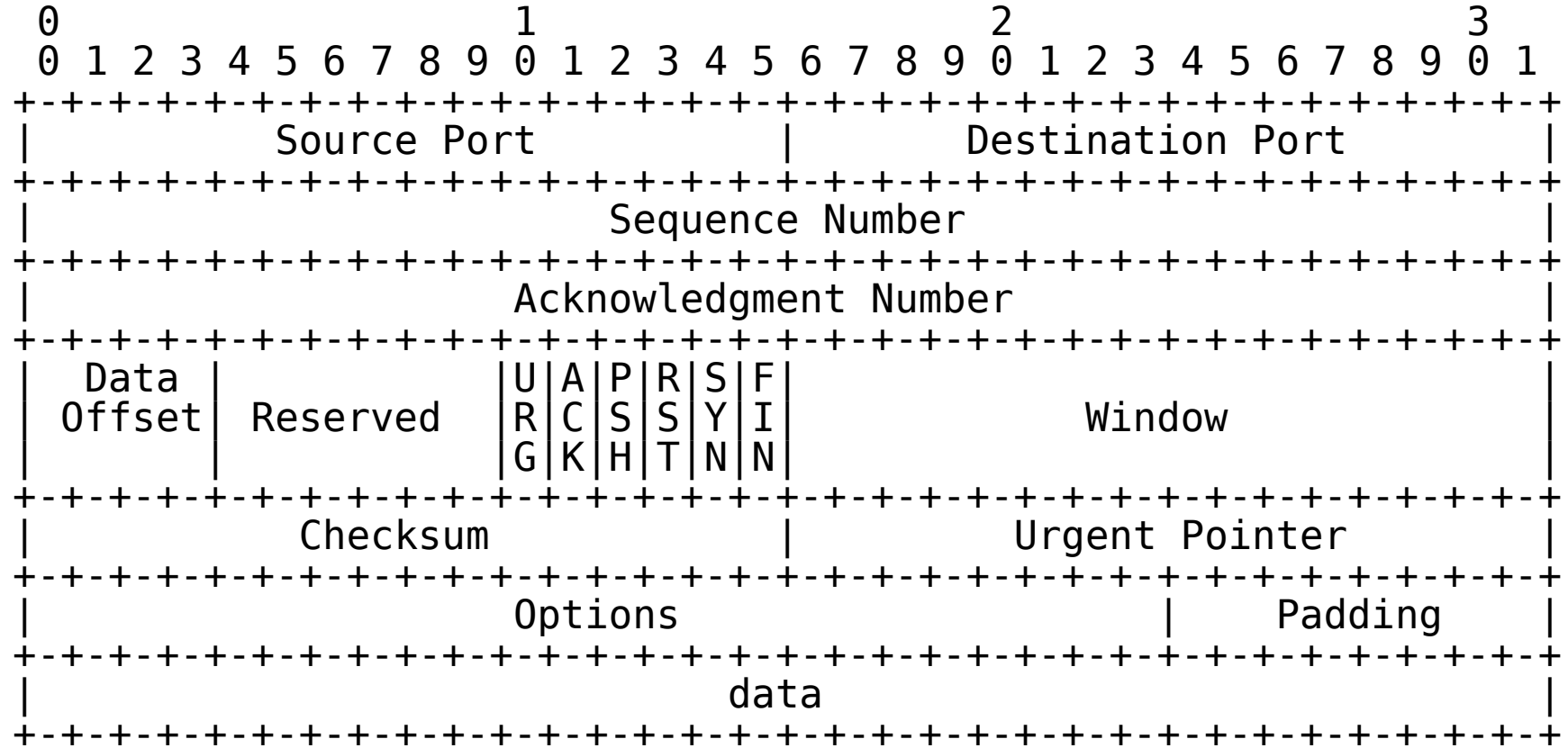
TCP implémente une sémantique par flots :



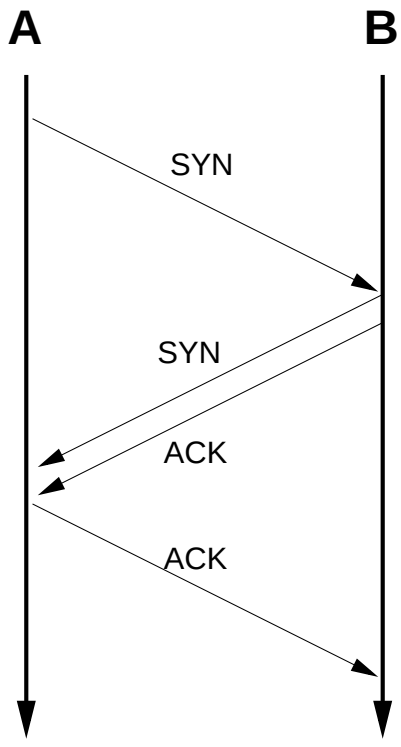
Un tampon est donc presque toujours nécessaire :

- lors de l'écriture, pour émettre toutes les données qui constituent un message (écritures partielles) ;
- lors de la lecture, pour accumuler un message entier qui peut arriver par morceaux.

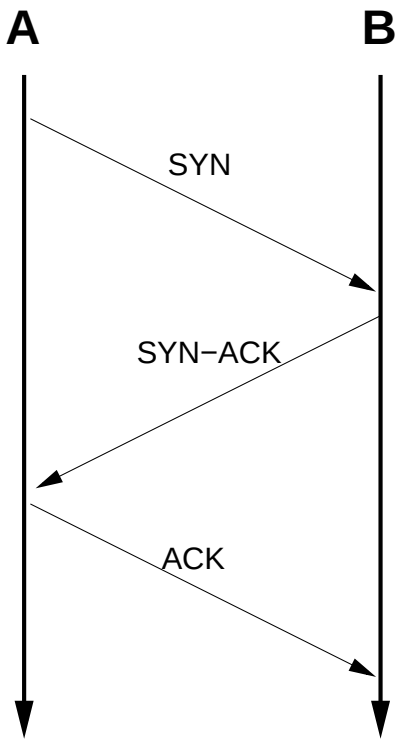
TCP : format des paquets



TCP : établissement de connexion

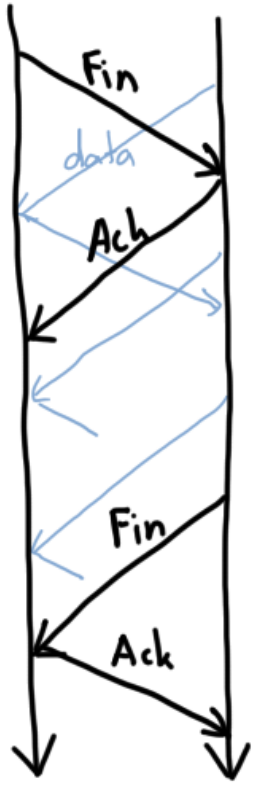


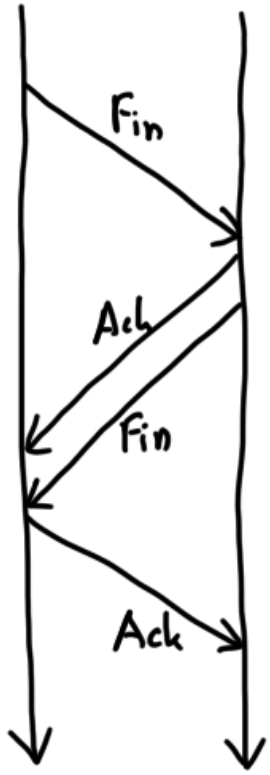
(a)

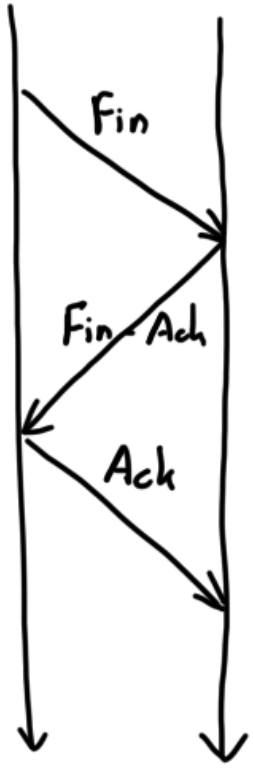


(b)

TCP : fin de connexion







TCP : transfert de données

- Segmentation et réassemblage ;
- fiabilité (acquittements, réémissions) ;
- contrôle de flot (ne pas surcharger le récepteur) ;
- contrôle de congestion (ne pas surcharger le réseau).

TCP : segmentation et reassemblage

Les octets d'un flot TCP sont numérotés.
Origine : ISN + 1 (ISN transmis dans SYN).

L'émetteur segmente les données transmises.

Le récepteur les réassemble :

- les remet dans l'ordre ;
- les concatène.

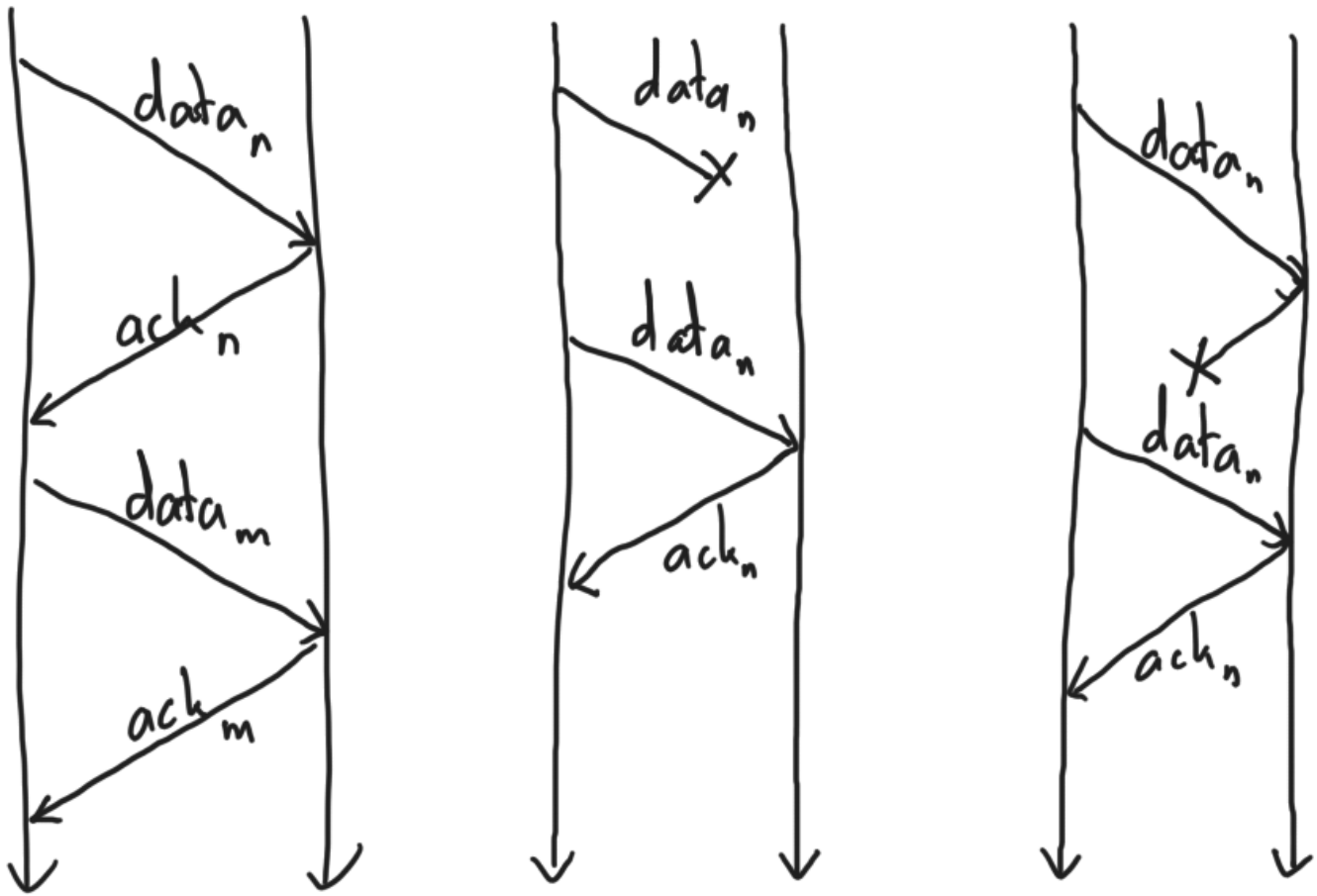
Digression : fiabilité

« Les données arrivent toujours »
$\longrightarrow$ pas implémentable.

« Les données arrivent ou on a une indication d'erreur »
$\longrightarrow$ pas implémentable.

Les données arrivent ou on a une indication d'erreur
ou les deux.

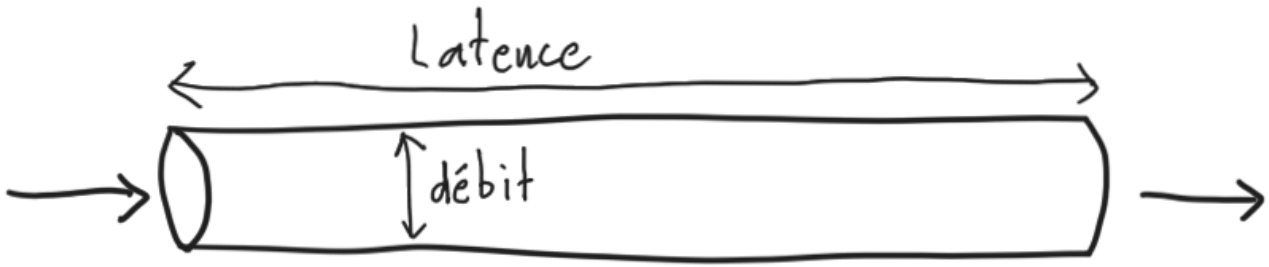
TCP : Acquittements et réémissions



Digression : débit et latence

Le débit est la quantité de données qu'on peut transmettre par unité de temps (ex : 100 Mbit/s).

La latence est le temps que les données mettent à arriver.
Le *round-trip time* (RTT) ou *ping* est le temps d'aller-retour (ex : 10 ms).



Débit et latence (2)

Le débit est la quantité de données qu'on peut transmettre par unité de temps (ex : 100 Mbit/s).
La latence est le temps que les données mettent à arriver.

On peut presque toujours augmenter le débit :

- une fibre optique a une bande passente presque infinie ;
- plusieurs fibres en parallèle.

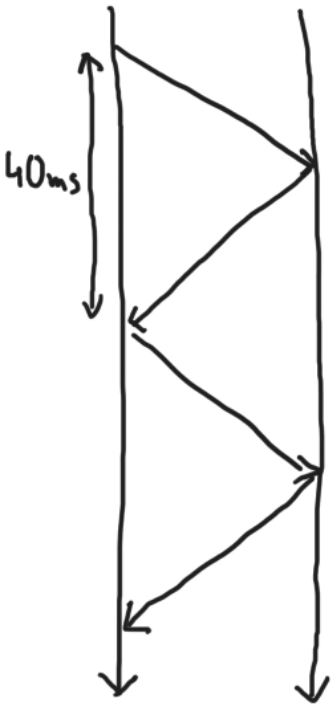
La latence est incompressible au delà d'un certain point :

- la vitesse de la lumière est glacialement lente.

Latence : conséquences

La vitesse de la lumière est glacialement lente : 30 cm/ns.

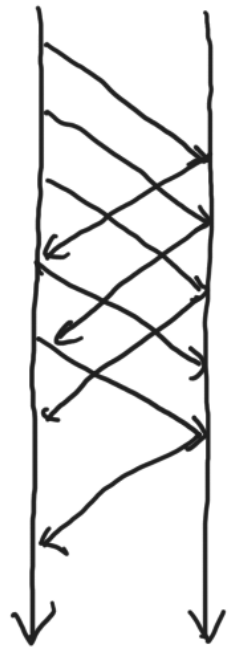
Paris-New York : 20 ms.



Paquet : 12 000 bits.

Un paquet toutes les 40 ms : débit limité à 300 kbit/s.

Même si on augmente le débit du lien !



Il faudra donc envoyer plusieurs paquets avant de recevoir un acquittement.
Combien ? C'est le problème du contrôle de flot et de congestion.

Contrôle de flot

On va envoyer plusieurs paquets avant de recevoir les acquittements.
Que se passe-t-il si le récepteur est lent
(ex. imprimante) ?

Les tampons du récepteur débordent. Les paquets sont réemis.
Il faut limiter le nombre de paquets émis pour qu'il ne déborde pas des tampons du récepteur. C'est le contrôle de flot.

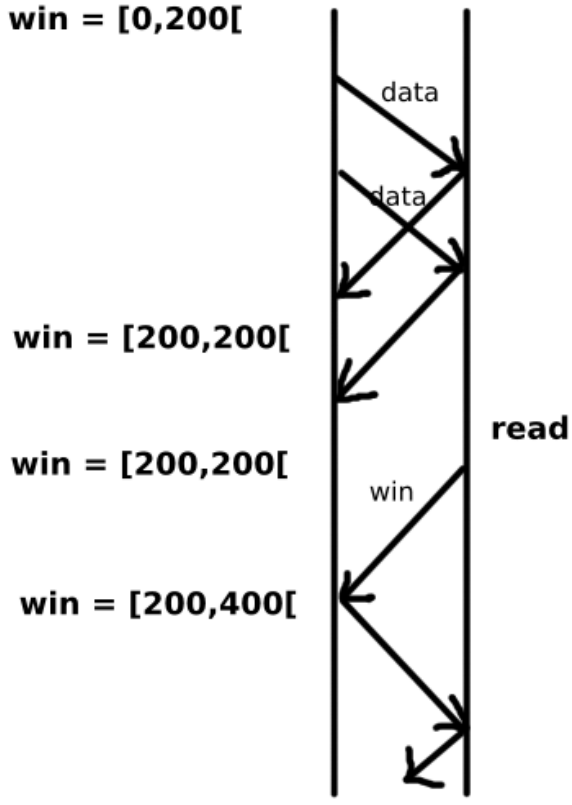
Contrôle de flot : fenêtre coulissante

Définition : la fenêtre du récepteur est l'ensemble des numéros de séquence que l'émetteur a le droit d'émettre.

Définie par deux variables maintenues par l'émetteur :

- bord gauche : le plus grand numéro de séquence aquitté Ack ;
- bord droit : communiqué explicitement par l'émetteur dans un segment Win.

Fenêtre coulissante : exemple



Contrôle de congestion

Initialement, Internet constitué de liens rapides (64 kbit/s), tous identiques.

En 1986, *Ford Aerospace* est interconnecté à l'Internet par un lien lent.

John Nagle constate que :

- aucun traffic utile ne passe ;
- les tampons des routeurs sont pleins de réémissions.

John Nagle, *On packet switches with infinite storage*, 1987.

Contrôle de congestion (2)

Dans un réseau qui mélange des liens rapides et des liens lents, une file de paquets s'accumule avant le goulot d'étranglement : congestion.

Comme ces paquets ne sont pas acquittés, ils sont réémis, ce qui empire la situation.

Il faut un mécanisme de contrôle de congestion.

Fenêtre de l'émetteur

TCP-Reno interprète chaque perte comme une indication de congestion.

Fenêtre de congestion :

- initialisée à 2 paquets ;
- croît à chaque acquittement ;
- décroît en cas de perte de paquet ;
- régie par deux algorithmes (*Slow start* et *congestion avoidance*.

Un paquet peut être émis s'il est dans les deux fenêtres.

Slow start

Algorithme MIMD :

- à chaque acquittement, $f := f + 1$ segment ;
- en cas de pertes, $f := f/2$ une fois par RTT.

Mène à

- décroissance exponentielle en cas de perte ;
- croissance exponentielle.

Congestion avoidance

Slow start est très agressif : au moins un paquet perdu par RTT.
Après le premier épisode de congestion, TCP passe à congestion avoidance :

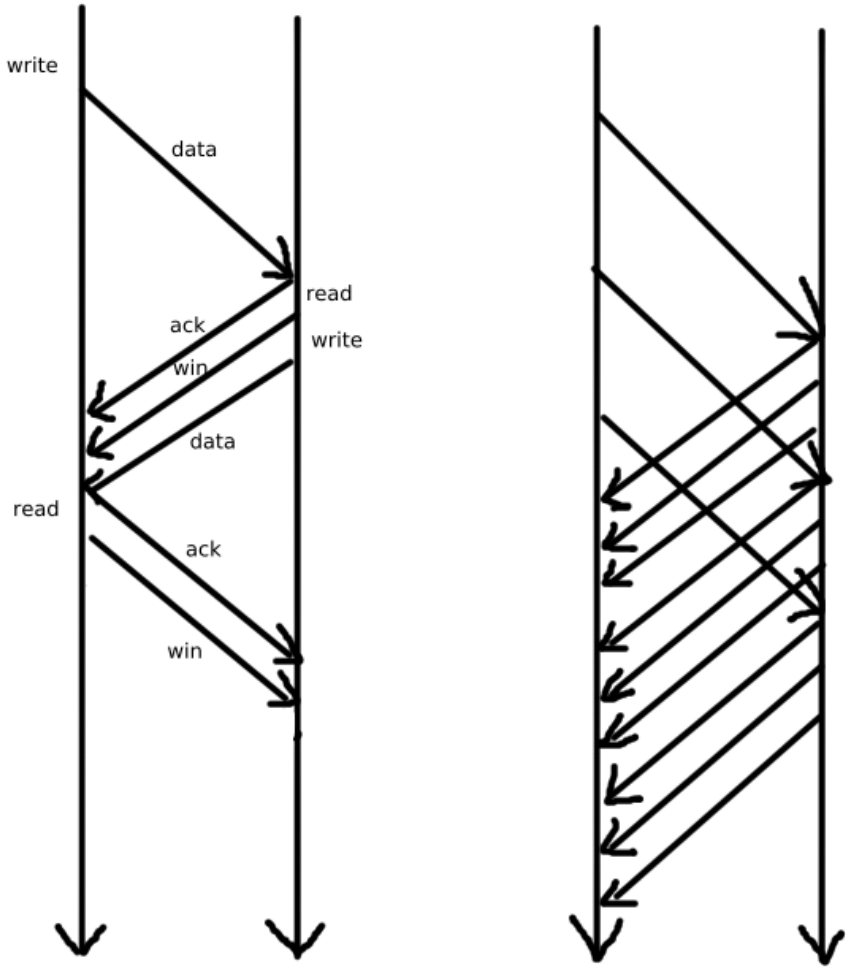
- en l'absence de pertes, $f := f + 1$, segment une fois par RTT ;
- en cas de pertes, $f := f/2$ une fois par RTT.

Optimisations

Au cours des années, on a ajouté à TCP-Reno plusieurs optimisations.

- algorithme de Nagle : exemple d'algorithme bien fait ;
- delayed Ack : exemple d'algorithme mal fait.

Algorithme de Nagle



Algorithme de Nagle

Lorsque l'application fait un `write` :

- pas de données en vol $\longrightarrow$ émis tout de suite ;
- suffisamment pour remplir un segment de taille maximale $\longrightarrow$ émis tout de suite ;
- sinon $\longrightarrow$ on attend.

Peut être éteint : `setsockopt(TCP_NODELAY)`.

Acquittements retardés

A priori, un acquittement pour chaque paquet de données.

Problème : laisons satellite avec asymétrie > 40.

Or, les acquittements TCP sont cumulatifs.

Solution très primitive :

- un acquittement sur deux est émis ;
- l'autre acquittement est retardé de 500 ms, émis si un autre paquet n'arrive pas.

Plein de problèmes :

- *timeout* fixé ;
- interatit mal avec Nagle ;
- ne peut pas être eteint !

Autres optimisations

- acquittements dupliqués ;
- *silly-window syndrome (SWS) avoidance* ;
- acquittements sélectifs ;
- ...

Congestion basée sur le délai

TCP-Reno réduit le fenêtre de congestion en cas de perte de paquets. C'est trop tard !

Or, une augmentation de la latence indique une augmentation de la longueur de file dans le routeur. Idée : ralentir le débit lorsque la latence augmente.

TCP-Vegas, BBR, GCC...

Problème : ces algorithmes sont moins agressifs que TCP-Reno, donc ils se font dominer par ce dernier.

Autres protocoles de couche transport

Il existe d'autres protocoles de couche transport :

- UDP-Lite : UDP sans détection d'erreurs ;
- DCCP : UDP plus contrôle de congestion ;
- SCTP : fiable, par messages, flots multiples.

Bloqués par les NAT ⟶ pas déployés.

Protocoles encapsulés dans UDP :

- RTP (vidéoconférence) ;
- µTP (BitTorrent) ;
- QUIC (HTTP/3) ;
- SCTP sur UDP.

Encapsulation dans UDP

Le service fourni par UDP est semblable à IP.
(Juste les numéros de ports en plus.)

On peut donc encapsuler un protocole de couche transport dans UDP au lieu de IP.

(Couches 7-4-3-3-2-1. Ou peut-être UDP est-il maintenant à la couche 3 ?)

Avantages :

- traverse les NAT ;
- implémentable avec l'API *sockets*.

Désavantages :

- 8 octets par paquet.