TCP : Implémentation du transfert fiable

TCP est un protocole qui implémente le transfert fiable dont on a parlé juste avant. Il comprend 3 phases,

- La phase de connexion qui utilise un three-way handshake
- Le transfert d'informations en utilisant des acquits comme vu précédemment
- La fermeture de connexion

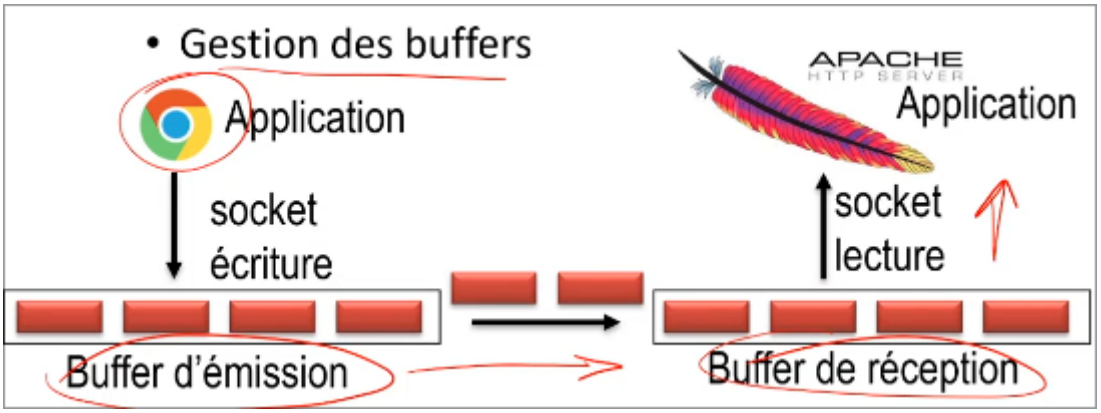
TCP fonctionne également en **unicast**, c'est-à-dire d'un destinataire à un autre et pas à un groupe de destinataire (pas multicast).

Une connexion dans TCP est identifié par quatre informations, le port source, le port de destination, l'IP source et l'IP destination.

TCP définit aussi un MSS (Maximum Segment Size) qui indique la taille maximum des données qui peuvent être dans un TPDU, cela dépends souvent de la couche réseau et de liaison utilisée, par exemple 1500 octets pour Ethernet.

Il y a par ailleurs une extension à TCP qui est utilisée dans la plupart des systèmes d'exploitations qui est le multipath TCP qui consiste à utiliser plusieurs accès réseaux simultanés pour transférer des données plus rapidement (par exemple utiliser la 4G et le Wifi en même temps).

Pour ne pas tout le temps envoyer des choses sur le réseau en permanence, TCP utilise des buffers. Ainsi, lorsque qu'une application veut envoyer des TPDU, il les place dans un buffer d'écriture, une fois plein, les informations sont envoyées. Lors de la réception, les données sont placées dans un buffer de destination, une fois toutes les données reçues, les données sont envoyées à l'application.



TPDU TCP

Un TPDU TCP est composé de plusieurs informations :

- Port source
- Port destination
- Numéro de séquence
- Numéro d'acquittement
- Taille de l'entête
- Des indicateurs (ACK pour acquis, SYN pour demander une connexion, FIN pour terminer une connexion, etc)
- La taille de la fenêtre glissante
- Le "checksum" pour vérifier les données
- Le pointeur de donnée urgente et les options, qui ne servent à rien ou sont facultatives
- Données

Ce système permet donc de faire de la communication bidirectionnelle étant donné qu'il est possible de mettre des données et un acquit dans un même TPDU. On considère en TCP que l'acquit est toujours le prochain numéro de séquence attendu.

Temporisateur TCP

TCP doit également définir un temporisateur, c'est-à-dire mettre un "timeout" au bout duquel, si aucun acquit a été reçu, le(s) TPDU sont considérés comme perdu et doivent être renvoyé.

Ce délai doit donc être plus grand que le RTT (Round-Trip Time) qui est le temps de faire un aller-retour entre un émetteur et une destination. Aussi, si le RTT est très variable, le délai du temporisateur sera plus grand.

La valeur du temporisateur est alors $ RTTmoyen + 4 * RTTvariation $.

Envois des acquits

Il y a plusieurs cas différents d'envois d'acquits en TCP, lors de la réception d'un nouveau TPDU :

- SI on reçoit un TPDU ayant le numéro de séquence attendu ET que toutes les données ont été acquittées jusque-là, ALORS on attend jusqu'à 500 ms ET si aucun TPDU n'arrive au bout de ce délai, ALORS on acquitte le TPDU

- SI on reçoit un TPDU ayant le numéro de séquence attendu MAIS que toutes les données n'ont pas été acquittées jusque-là, ALORS on envoie un acquit pour tous
- SI on reçoit un TPDU ayant un numéro de séquence plus grand que prévu, ALORS on re-duplique l'acquit précédent pour demander le TPDU manquant
- SI on reçoit un TPDU couvrant un trou dans la séquence, ALORS on envoie acquit pour demander le prochain TPDU de la séquence

Fast retransmit

Une amélioration de TCP est le **Fast Retransmit** qui permet de ne pas avoir à attendre le timeout du temporisateur pour renvoyer un ou des TPDU.

Pour ce faire, le destinataire va envoyer des acquits pour tous les TPDU de la séquence, même ceux qui sont perdus. Ensuite, le destinataire va envoyer trois acquits identiques pour le TPDU perdu.

La réception de ces trois acquits est vue comme une demande de ré-envois de ces données par l'émetteur qui n'a ainsi pas besoin d'utiliser son temporisateur dans ce cas. Cela permet donc d'aller beaucoup plus vite.

Gestion des pertes avec TCP

TCP retient uniquement les TPDU qui arrivent en séquence (donc avec les numéros de séquence attendus à chaque fois).

Mais TCP sauvegarde tout de même les TPDU qui arrive hors séquence afin de ne pas avoir à les redemander plus tard.

La fast-retransmit vu plus tôt permet à TCP d'aller plus vite pour demander les données perdues à l'émetteur.

Algorithme de Nagle pour l'émission de TPDU

Il serait fort peu pratique d'émettre des TPDU pour chaque petite donnée, car avoir beaucoup de petit TPDU causerait des problèmes de congestion du réseau.

L'idée est alors de combiner plusieurs TPDU dans un seul, plus gros TPDU. Le fonctionnement de cet algorithme est celui-ci :

1. Le premier octet est envoyé immédiatement
2. Tant que l'accusé de réception n'est pas reçu, on accumule toutes les données suivantes dans un seul TPDU (tampon ou buffer). Lorsque l'acquittent arrive, on envoie le TPDU.
3. On répète la deuxième étape.

Contrôle de flux et de la fenêtre

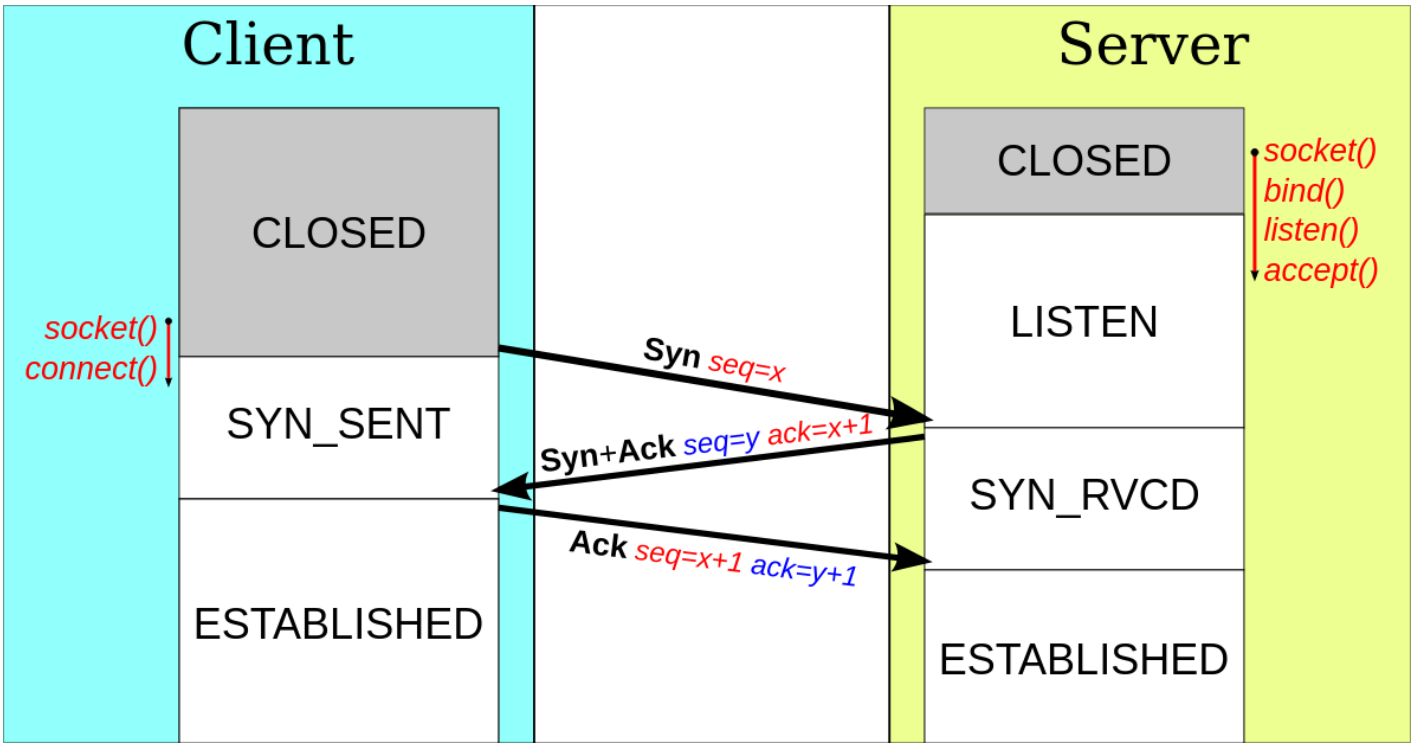
Le destinataire doit indiquer la taille de sa fenêtre (qui symbolise sa capacité de traitement) au fur et à mesure afin de ne pas être submergé de données.

L'émetteur de son côté est obligé d'attendre la réception d'un acquit lorsque la fenêtre est vide avant de recommencer à transmettre.

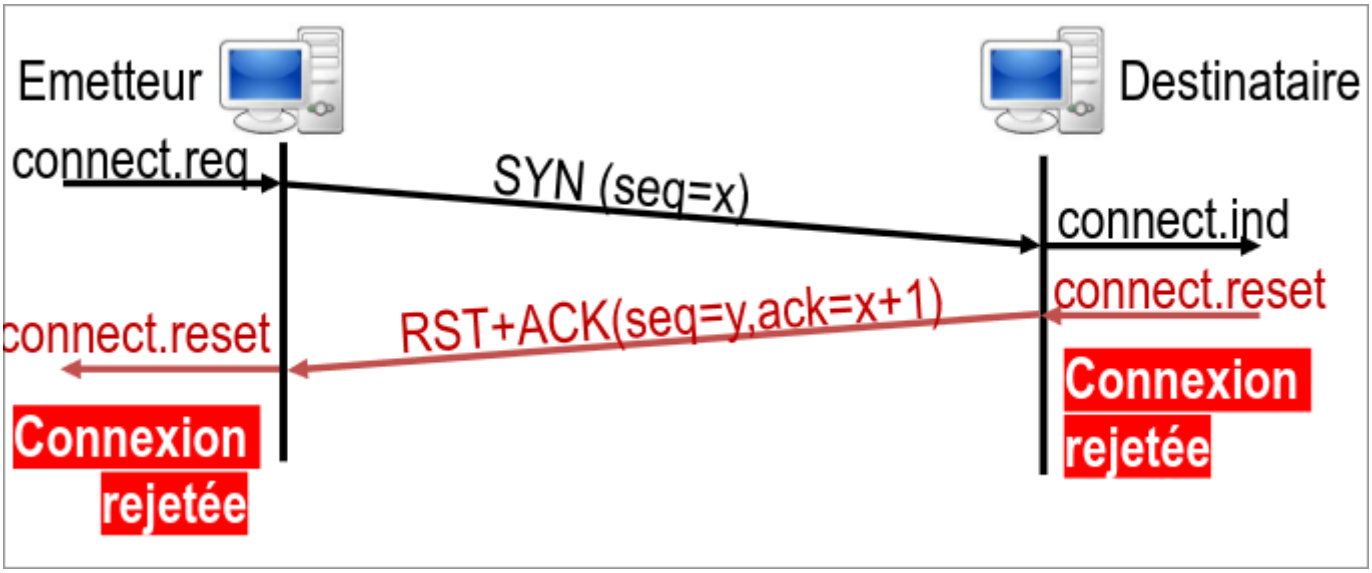
Connexion et déconnexion TCP

Une requête de connexion en TCP se fait avec le flag `SYN`. Le serveur peut ensuite soit accepter en utilisant les flags `SYN` et `ACK`, ou refuser avec `RST` et `ACK`. Si la connexion est acceptée par le serveur, le client envois un `ACK` pour signaler qu'il est prêt à commencer l'échange de données, ou un `RST` et `ACK` pour refuser et annuler la connexion.

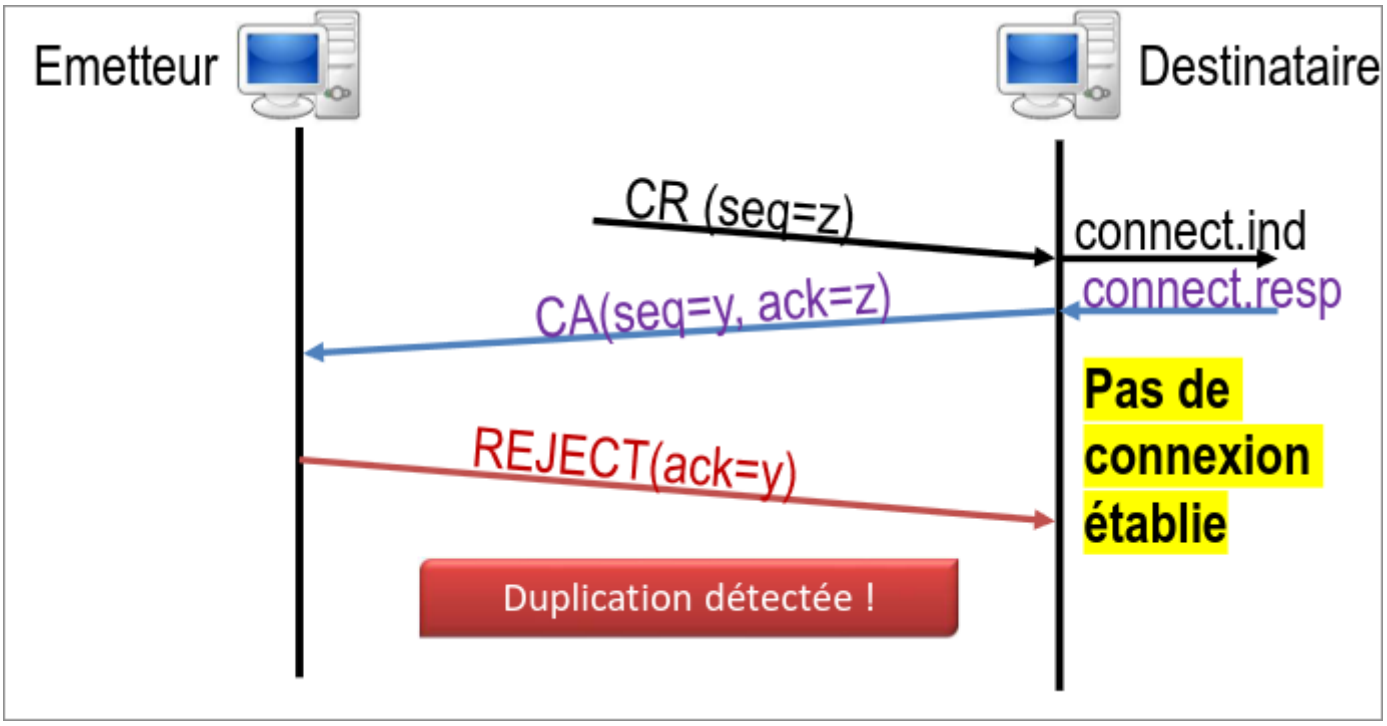
Voici ce qu'il se passe lorsque la connexion est acceptée par le serveur :



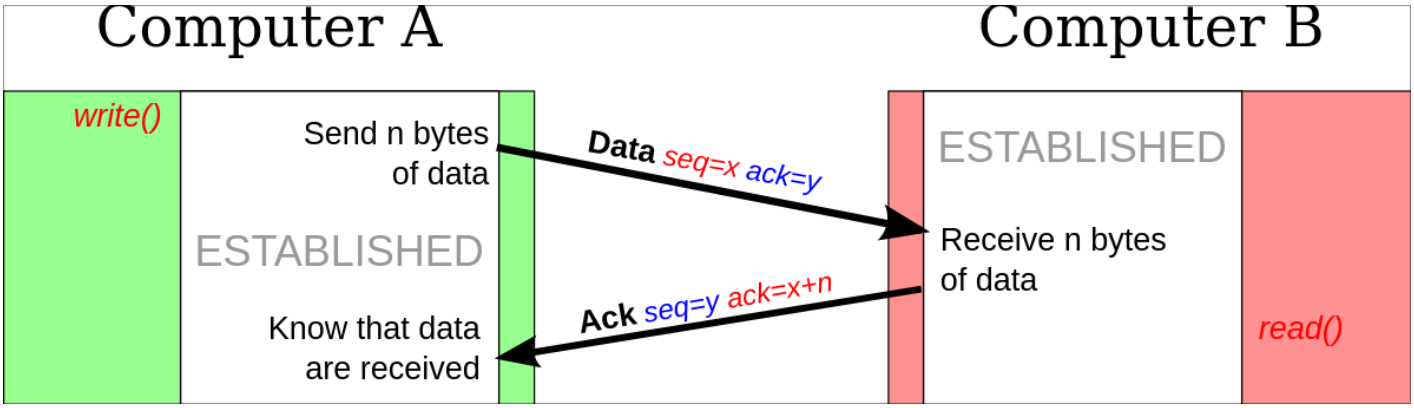
Voici ce qu'il se passe lorsque la connexion est refusée par le serveur :



Et voici ce qu'il se passe lorsque la connexion est invalide et est refusée par le client (compter que REJECT serait en TCP `SYN+ACK`) :

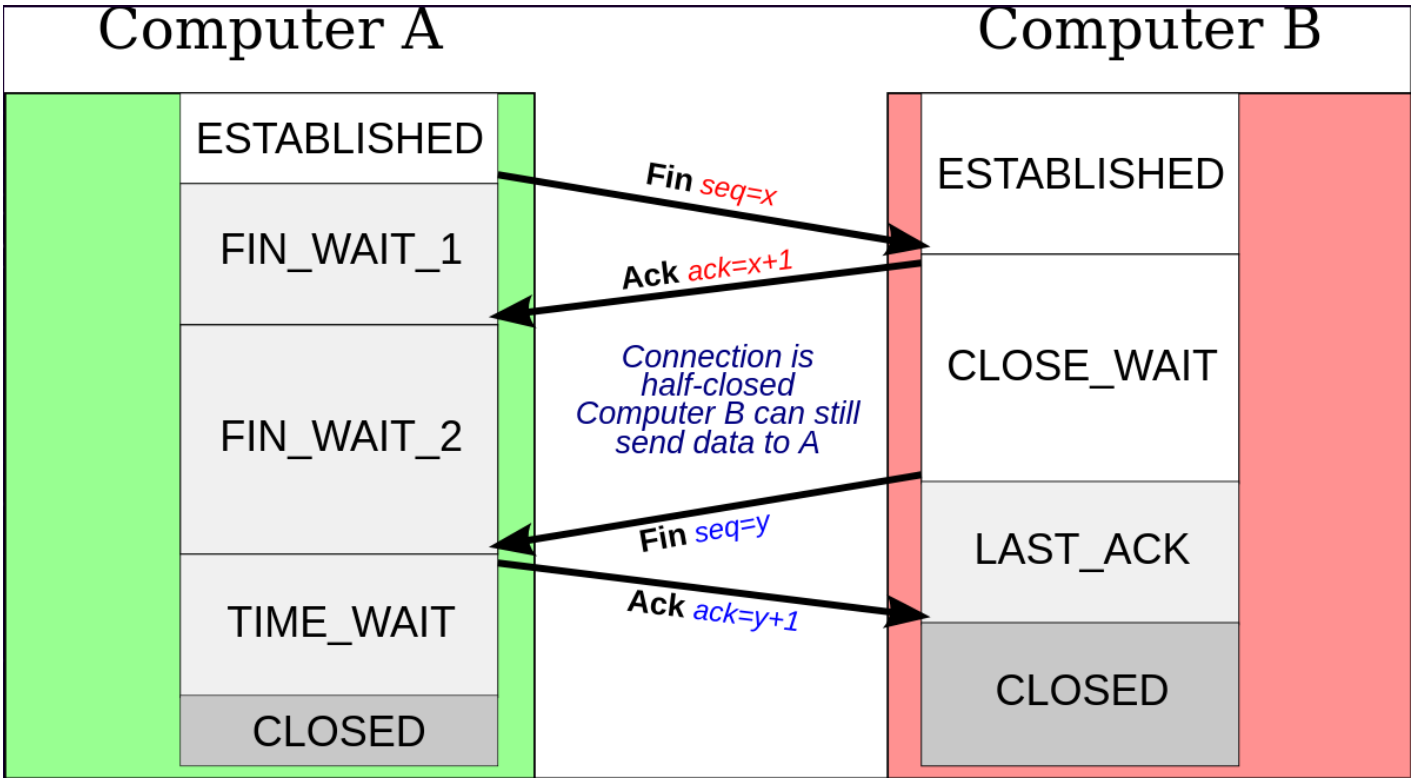


Une fois la connexion établie, un échange de donnée peut donc s'opérer :



Une fois l'échange de donnée terminé, la connexion peut être fermée correctement en utilisant une méthode similaire au three-way handshake utilisé pour la connexion. On envoie un TPDU avec un flag `FIN` pour demander la déconnexion, l'autre partie renvois donc un `ACK` transfert également quelques dernières données, puis envois un `FIN` à son tour qui a une réponse par un `ACK`. La connexion est alors fermée.

Il est aussi possible d'avoir une connexion abrupte, si un `FIN` est envoyé sans attendre d'acquit. Vous pouvez avoir plus de détail en lisant la partie plus tôt sur la déconnexion dans le transfert fiable.



Contrôle de la congestion

Tous les réseaux sur internet ne sont pas égaux, certains sont donc beaucoup plus lents que d'autres. Pour éviter de surcharger ces réseaux avec beaucoup de données, il est donc important de mettre en place un système permettant de limiter cette congestion.

Pour cela, l'émetteur va retenir une "fenêtre de congestion" (qui n'a rien à voir avec la fenêtre glissante).

Cette fenêtre de congestion indique la quantité de donnée qui peut être transmise, elle est mesurée en MSS (maximum segment size, c'est-à-dire la taille maximale d'un TPDU). Au départ, elle est à 1, et à chaque `ACK` reçu, elle augmente de 1. Cette donnée va donc grandir de manière exponentielle. Cette phase est appelée le **slow-start**.

Ensuite, une fois que la fenêtre atteint un certain maximum prédéfini, elle va ensuite grandir de manière linéaire en augmentant de 1 à chaque RTT (Round-Trip Time, donc il faudra attendre que toutes les données envoyées soient acquittées). Cette phase est appelée le **AIMD** (Adaptative Increase Multiplicative Decrease).

Ce qui est fait lorsque des données sont perdues dépends de comment on sait que les données sont perdues :

- Soit, on reçoit un **Fast-Retransmit** (3 acquits dupliqués), la congestion est alors définie comme légère et on définit le maximum de la fenêtre à la valeur actuelle de la fenêtre divisée par 2. Et on définit la valeur de la fenêtre au seuil. On reprend alors en mode linéaire (AIMD).

- Soit, il y a un **timeout du temporisateur**, la congestion est alors définie comme forte et on définit le maximum de la fenêtre à la valeur actuelle de la fenêtre divisée par 2. Et on met la valeur de la fenêtre à un et on recommence en slow-start (exponentiel).

Revision #1
Created 29 February 2024 14:22:31 by SnowCode
Updated 29 February 2024 14:32:49 by SnowCode