RFC 7323 : TCP Extensions for High Performance

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 21 septembre 2014

Date de publication du RFC : Septembre 2014

https://www.bortzmeyer.org/7323.html

L'algorithme originel de TCP rendait ce protocole de transport trop prudent et n'utilisant pas assez les réseaux, notamment ceux à forte latence <https://www.bortzmeyer.org/latence.html>. Après quelques essais, le RFC 1323[1], publié en 1992, a permis à TCP de fonctionner correctement sur une bien plus grande variété de réseaux, et jusqu'à aujourd'hui. Il est désormais remplacé par ce nouveau RFC 7323 qui, après une longue genèse, représente l'état de l'art en matière de performances TCP. Ce nouveau RFC est une lecture indispensable pour les fans de TCP ou tout simplement pour ceux qui veulent comprendre en détail ce protocole.

Avant le RFC 1323, TCP (normalisé dans le RFC 793 en 1981) se comportait très bien sur les réseaux locaux, ainsi que sur les réseaux distants à faible débit, comme ce qu'on avait sur un modem. Mais il était beaucoup moins satisfaisant sur les réseaux à forte latence <https://www.bortzmeyer.org/latence.html> et forte capacité <https://www.bortzmeyer.org/capacite.html>, les réseaux à fort **BDP** où BDP signifie "*Bandwitdh-Delay Product*". Si la capacité est faible ou la latence faible, pas de problèmes. Si leur produit dépasse une certaine valeur, TCP n'était pas capable de remplir la **fenêtre** et ses performances restaient en deçà du maximum théorique du réseau.

La section 1 décrit ce problème. TCP avait été conçu (et avec succès) pour tourner sur des réseaux très disparates, et pour s'adapter automatiquement à leurs caractéristiques (taux de perte, latence, taux de duplication...) À l'époque du RFC 1323, TCP tournait en production sur des réseaux dont les capacités allaient de 100 b/s à 10 Mb/s et cette plage s'est nettement élargie depuis. Existe-t-il une limite au débit de TCP, au-delà de laquelle il ne servirait à rien d'accélérer encore les réseaux ? La question n'a pas de réponse simple.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc1323.txt

La caractéristique importante du réseau n'est en effet pas la capacité <https://www.bortzmeyer.org/capacite.html> mais le produit de la capacité et de la latence, le BDP cité plus haut. C'est cette caractéristique qui indique la taille du tuyau que TCP doit remplir, la capacité étant le « diamètre » du tuyau et la latence sa « longueur ». Si la capacité croît beaucoup, au rythme des progrès techniques, la latence est bloquée par la finitude de la vitesse de la lumière et la seule façon de l'améliorer est de raccourcir les câbles. Donc, un gros BDP oblige TCP à avoir davantage de données « en transit », envoyées, mais n'ayant pas encore fait l'objet d'un accusé de réception, ce qui implique des tampons d'entrée/sortie de grande taille mais qui implique aussi la possibilité de garder trace de grands nombres (par exemple le nombre d'octets en transit), donc d'avoir des compteurs de taille suffisante. Ces liaisons Internet avec un fort BDP sont parfois surnommées les « éléphants » de l'anglais LFN ("*Long Fat Network*").

Un exemple typique d'éléphant est une liaison satellite, avec sa capacité souvent respectable mais sa latence terrible, due à la nécessite d'un aller-retour avec l'orbite géostationnaire. À l'époque du RFC 1123, le BDP de ces liaisons était d'environ 1 Mbit soit 100 segments TCP de 1 200 octets chacun. Si une mise en œuvre de TCP se limitait à 50 segments envoyés avant de recevoir un accusé de réception, elle n'utiliserait que la moitié de la capacité disponible. Et les liaisons terrestres peuvent être des éléphants aussi. Un lien transcontinental aux États-Unis a une latence de 30 ms, ce qui, à 45 Mb/s, fait également un BDP de 1 Mbit.

Qu'est-ce qui empêchait TCP de tirer profit de ces éléphants ? Trois points :

— La taille de la fenêtre n'est stockée par défaut que sur 16 bits, ne permettant pas de fenêtre plus grande que 65 535 octets. Ce problème est résolu par le RFC 1323 avec l'introduction du "*window scaling*".
— La récupération était trop longue en cas de perte de paquets. Les premiers TCP, dès qu'un paquet était perdu, attendaient de vider complètement le pipeline, puis repartaient de zéro, comme pour une connexion TCP neuve. En 1990, l'algorithme de TCP avait été modifié pour permettre un redémarrage plus rapide, tant qu'on ne perdait qu'un seul paquet par fenêtre TCP. Mais, avec des fenêtres plus grandes, cette probabilité de perte augmente. Les accusés de réception de TCP étant cumulatifs, une perte de paquet survenant au début de la fenêtre peut faire tout perdre. La solution a été une option d'accusés de réception sélectifs (SACK pour "*Selective ACKnowledgment*"). Ce point n'a pas été traité dans le RFC 1323 mais dans le RFC 2018.

Un autre problème à considérer est la fiabilité. Si on utilise TCP, c'est pour avoir certaines garanties : que tous les octets émis seront reçus, dans le même ordre, etc. Est-ce que le passage à de plus hautes performances menace ces garanties ? Par exemple, avec des fenêtres plus grandes, la probabilité qu'un paquet ancien, appartenant à une précédente connexion, lorsqu'il finit par arriver, tombe dans la fenêtre courante, cette probabilité est plus élevée. Dans ces conditions, les données seraient corrompues. La principale protection de TCP contre cet accident est la notion de MSL ("*Maximum Segment Lifetime*"), le temps qu'un segment peut traîner sur l'Internet. Il ne faut pas réutiliser des numéros de séquence avant qu'une durée supérieure ou égale à la MSL se soit écoulée. Ce numéro ne faisant que 32 bits, cela peut être délicat, surtout aux débits élevés (même sans fenêtres agrandies). La MSL est généralement prise à deux minutes et, à seulement 1 Gb/s, les numéros de séquence ne durent que dix-sept secondes. Or, aucun mécanisme sur l'Internet ne garantit le respect de la MSL. Un vieux paquet ne sera pas jeté. D'où l'utilisation par notre RFC 7323 de l'option "*Timestamps*" pour détecter les segments trop anciens et se protéger donc contre la réutilisation des numéros de séquence TCP (solution PAWS, en section 5).

À noter que ces mécanismes sont conçus pour les réseaux à fort BDP. Sur des réseaux à faible BDP, il peut être intéressant de les débrayer, manuellement ou automatiquement.

Reste que les solutions proposées dans ce RFC dépendent des options TCP. Pour certains protocoles, par exemple IP, certaines options ont du mal à passer à travers le réseau (section 1.3 de notre RFC). TCP semble mieux placé de ce point de vue (il est mentionné à la fin de mon article sur les options IP <https://www.bortzmeyer.org/options-interdites.html>). On peut consulter à ce sujet « "*Measuring Interactions Between Transport Protocols and Middleboxes*" <http://www.icir.

net/tbit/tbit-Aug2004.pdf> » et « "*Measuring the Evolution of Transport Protocols in the Internet*" <http://icir.net/floyd/papers/TCPevolution-Mar2005.pdf> ».

La section 2 de notre RFC présente la première option qui avait été normalisée pour améliorer les performances de TCP sur les liens à fort BDP ("*Bandwidth-Delay Product*"), le "*window scaling*". L'idée de base est très simple : 16 bits pour indiquer la taille de la fenêtre, c'est trop peu, on va donc appliquer un facteur (indiqué dans une option TCP) au nombre décrit par ces 16 bits. À noter que, comme les options ne sont envoyées qu'au début de la connexion TCP, le facteur est constant (la fenêtre elle-même étant dynamique).

L'option "*Window Scale*" comprend trois champs : Type, Longueur et Valeur. Le type vaut 3 et est enregistré dans le registre des options <https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1>, la longueur est forcément de 3 (trois octets en tout) et la valeur est un octet qui indique de combien de bits on va décaler la taille de la fenêtre. Une valeur de 0 indique pas de décalage, donc un facteur de 1 (une telle valeur n'est pas inutile car elle sert à indiquer au pair TCP qu'on sait gérer le "*window scaling*"). Une valeur de 1 indique qu'on double la taille de la fenêtre pour connaître la vraie valeur, etc. Voici un exemple vu par Wireshark :

```
Transmission Control Protocol, Src Port: 51336 (51336), Dst Port: 4332 (4332), Seq: 0, Len: 0
...
   Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
...
        Window scale: 5 (multiply by 32)
            Kind: Window Scale (3)
            Length: 3
            Shift count: 5
```

Et, quelques paquets plus loin, on voit bien le facteur d'échelle appliqué (32, soit 2{}5). Le champ indiquant la longueur de la fenêtre vaut 728 octets mais il faut en fait lire 23 296 octets :

```
    Window size value: 728
    [Calculated window size: 23296]
    [Window size scaling factor: 32]
```

(À noter que je parlais aussi de cette option à la fin de l'article sur le RFC 793.) Sur Linux, cette option peut s'activer ou se désactiver avec le paramètre sysctl `net.ipv4.tcp_window_scaling` (c'est parfois nécessaire de la désactiver dans certains réseaux bogués qui bloquent les paquets TCP contenant des options inconnues d'eux).

Autre option normalisée ici, la meilleure mesure du RTT par l'option "*Timestamps*", en section 3. La mesure du RTT est cruciale pour TCP, pour éviter des accidents comme la congestion brutale décrite dans le RFC 896. Si TCP ne mesure qu'un seul paquet par fenêtre, les résultats seront mauvais pour les grandes fenêtres, par simple problème d'échantillonage (critère de Nyquist).

L'option "*Timestamps*" a le type 8, une longueur de 10, et deux champs de quatre octets, l'heure qu'il était au moment de l'envoi et l'heure lue dans le paquet pour lequel on accuse réception (cette valeur n'a donc de sens que si le paquet a le bit `ACK`). L'« heure » n'est pas forcément celle de l'horloge au mur (puisque, de toute façon, on n'utilisera que des différences), l'important est qu'elle avance à peu près au même rythme. En fait, il est même recommandé que l'horloge ne soit pas directement celle de la machine, pour éviter de donner une information (la machine est-elle à l'heure) à un éventuel observateur indiscret. La section 7.1 recommande d'ajouter à l'horloge de la machine un décalage spécifique à chaque connexion, et tiré au hasard au début de la connexion.

Attention, il n'y a aucune raison qu'on ait le même nombre de paquets dans les deux sens. On peut voir un pair TCP envoyer deux paquets et le récepteur ne faire qu'un seul paquet d'accusé de réception. Dans ce cas, ledit récepteur devra renvoyer le temps du paquet le plus ancien. Toujours avec Wireshark, cela donne :

```
Transmission Control Protocol, Src Port: 4332 (4332), Dst Port: 51336 (51336), Seq: 0, Ack: 1, Len: 0
...
   Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
...
        Timestamps: TSval 2830995292, TSecr 27654541
            Kind: Timestamp (8)
            Length: 10
            Timestamp value: 2830995292
            Timestamp echo reply: 27654541
```

Et, dans le paquet suivant de la même direction, les compteurs ont augmenté :

```
        Timestamps: TSval 2830995566, TSecr 27654569
            Kind: Timestamp (8)
            Length: 10
            Timestamp value: 2830995566
            Timestamp echo reply: 27654569
```

Ici, il s'agissait d'une communication entre deux machines Linux. La génération des estampilles temporelles dans les options TCP est contrôlée par la variable sysctl `net.ipv4.tcp_timestamps` (documentée, comme les autres, dans le fichier `Documentation/networking/ip-sysctl.txt` des sources du noyau). Par exemple :

```
% sysctl net.ipv4.tcp_timestamps
net.ipv4.tcp_timestamps = 1
```

Cela signifie que cette option est activée sur cette machine (0 = désactivée).

Cette option d'estampillage temporel est utilisée dans PAWS (présenté plus loin) mais aussi dans d'autres systèmes comme ceux du RFC 3522 ou du RFC 4015.

La section 4 décrit l'utilisation des estampilles temporelles pour mesurer le RTT des paquets, ce qui sert à TCP à déterminer le RTO ("*Retransmission TimeOut*"), le délai au bout duquel TCP s'impatiente de ne pas avoir eu d'accusé de réception et réémet. Voir à ce sujet le RFC 6298, pour savoir tout de ce calcul du RTO, et aussi le papier « "*On Estimating End-to-End Network Path Properties*" <`http://aciri.org/mallman/papers/estimation-la.pdf`> ».

La section 5 présente le mécanisme PAWS ("*Protection Against Wrapped Sequence numbers*"), qui sert à lutter contre les vieux segments TCP qui arriveraient tard et avec, par malchance, un numéro de séquence qui a été réutilisé depuis et est donc considéré comme valide. Les numéros de séquence étant stockés sur 32 bits seulement, la probabilité d'un tel accident augmente avec la capacité des réseaux. PAWS se sert de la même option "*Timestamps*" qui a été présentée plus haut. L'idée est que si un segment TCP arrive avec une estampille temporelle trop ancienne, par rapport à celles reçues récemment, on peut le jeter sans remords. Comme pour tous les usages de l'option "*Timestamps*", il ne nécessite **pas** de synchronisation d'horloges entre les deux pairs TCP car les comparaisons se font toujours entre les estampilles mises par une même machine.

Quels sont les changements depuis le RFC 1323 (voir l'annexe H) ? D'abord, une partie du texte a été supprimée, celle consacrée à la discussion des mérites des différentes options. Si on veut lire cette discussion, il faut reprendre le RFC 1323.

Ensuite, de nombreux changements importants ont été apportés. Je ne vais pas les lister tous ici mais, par exemple, la section 3.2 a été très enrichie pour mieux préciser l'utilisation des estampilles temporelles (trop floue précédémment), l'algorithme de sélection de l'estampille dans la section 3.4 du RFC 1323 a été corrigé (deux cas n'étaient pas traités), le cas des paquets TCP `RST` ("*ReSeT*" d'une connexion) a été décrit, la discussion sur la MSS a été déplacée dans le RFC 6691, etc.

Nouveauté de ce RFC (le RFC 1323 clamait qu'il ne se préoccupait pas du sujet), la section 7, sur la sécurité. Ouvrir la fenêtre TCP pour augmenter les performances, c'est bien. Mais cela ouvre également la voie à des attaques où un méchant tente de glisser un paquet dans la séquence des paquets TCP. Normalement, un attaquant situé en dehors du chemin des paquets, qui ne peut donc pas les observer, doit, s'il veut réussir cette injection, deviner le numéro de séquence (RFC 5961). Mais plus la fenêtre est grande et plus c'est facile (il n'a pas besoin de deviner le numéro exact, juste de deviner un numéro qui est dans la fenêtre). Il faut donc mettre en rapport le gain de performances avec le risque d'accepter de faux paquets. PAWS protège partiellement contre ces attaques mais en permet de nouvelles (par exemple l'injection d'un paquet ayant une estampille dans le futur permettrait, si ce paquet est accepté, de faire rejeter les vrais paquets comme étant trop anciens).

Les fanas de programmation et de placement des bits dans la mémoire liront avec plaisir l'annexe A, qui recommande un certain arrangement des options dans le paquet TCP : en mettant deux options vides (`NOP`) avant l'option "*Timestamp*", on obtient le meilleur alignement en mémoire pour une machine 32-bits.