UNIVERSITÉ DE STRASBOURG

Les différentes implémentations de TCP

Philippe Pittoli

31 octobre 2013

1 LES DIFFÉRENCES ENTRE LES IMPLÉMENTATIONS HISTORIQUES

1.1 DÉFINITIONS

Avant de commencer, quelques éléments de terminologie :
- MSS (maximum segment size), la taille maximum d'un segment (entête TCP + données) ;
- awnd (advertised window), fenêtre de réception ;
- cwnd (congestion window), fenêtre de congestion ;
- swnd (sending window), fenêtre d'émission (= min (cwnd, awnd)) ;
- RTO (retransmit timeout), temps de transmission maximal (souvent 2 fois le RTT) ;
- RTT (roundtrip time), durée d'envoi puis de réception d'acquittement d'un paquet ;
- ssthresh (slow start threshold), taille maximum d'une fenêtre d'émission en slow start ;
- LFN (long fat network), réseau avec une grande bande passante mais aussi un long délai de réponse ;

1.1.1 SLOW START

Lors du démarrage d'une connexion, nous ne connaissons pas le lien qu'il y a entre nous et le destinataire, donc on va progressivement augmenter le nombre de paquets qu'on envoie. Le but est donc de trouver la bande passante disponible, et utiliser toutes les ressources à disposition. On va alors utiliser une fenêtre d'émission (swnd), qui représente un nombre de paquets à envoyer sans attendre d'acquittement. Cette fenêtre grandira au fur et à mesure que nous recevons des acquittements pour les paquets envoyés (chaque ACK fera augmenter cette fenêtre d'un paquet).

1.1.2 Congestion Avoidance

Au dela d'une certaine limite de valeur de cwnd (slow start threshold, ssthresh), TCP passe en mode d'évitement de congestion. À partir de là, la valeur de cwnd augmente de façon linéaire et donc bien plus lentement qu'en slow start : cwnd s'incrémente de un MSS (= un paquet) à chaque RTT. Dans ce mode de fonctionnement, l'algorithme détecte aussi rapidement que possible la perte d'un paquet : si nous recevons trois fois le ACK même paquet, on n'attend pas la fin d'un timeout pour réagir. En réaction à cette perte, on fait descendre la valeur de ssthresh ainsi que cwnd (on repasse éventuellement en mode de Slow Start). On utilise la technique de Fast Retransmit pour renvoyer rapidement les paquets perdus.

1.1.3 Fast Retransmit

Comme expliqué plus tôt, on passe par cet algorithme en cas de détection de perte de segment(s) lorsque nous sommes en mode « Congestion Avoidance ». Un ACK dupliqué s'explique par la manière dont les segments sont traités à la réception. En effet, si nous recevons un segment TCP qui n'est pas dans l'ordre attendu, on doit envoyer un ACK avec une valeur égale au numéro de segment qui était attendu. S'il y a une perte de segment, le destinataire n'enverra plus que des ACK avec le numéro du segment perdu. On peut donc palier à cette perte rapidement (après 3 ACK dupliqués généralement), sans attendre le timeout.

1.1.4 Fast Recovery

Plutôt que repasser en mode Slow Start lors d'une duplication de ACK (et après un passage par le mode Fast Retransmit), nous renvoyons le segment perdu et on attend un ACK pour toute la fenêtre transmise précédemment avant de retourner en mode Congestion Avoidance. Si on atteint le timeout, on repart en mode Slow Start. Grâce à cette technique nous évitons de baisser le débit d'une façon trop brutale.

L'ensemble des algorithmes présentés ci-dessous se basent sur ces techniques, ou une partie.

1.2 TAHOE

La première implémentation d'algorithme d'évitement de congestion de TCP est TCP Tahoe. C'est la plus simple et la moins efficace (tous types de problème confondus). Cette version utilise un système de slow start, avec une valeur initiale de cwnd à 1 et une valeur de cwnd maximum de ssthresh (avec une valeur par défaut de 65535). La première fois qu'on effectue du slow start on arrive à une perte de paquet, dans ce cas la valeur de cwnd courante sera notre nouveau ssthresh puis on remet la valeur de cwnd à 1. Une fois ssthresh atteint, on entre en Congestion Avoidance. À partir de là la valeur de cwnd augmente de façon linéaire et donc plus lentement qu'en Slow Start. Lorsqu'on reçoit trois fois le même ACK, il y a une congestion, on n'attend pas le timeout avant de renvoyer le paquet perdu (fast retransmit).

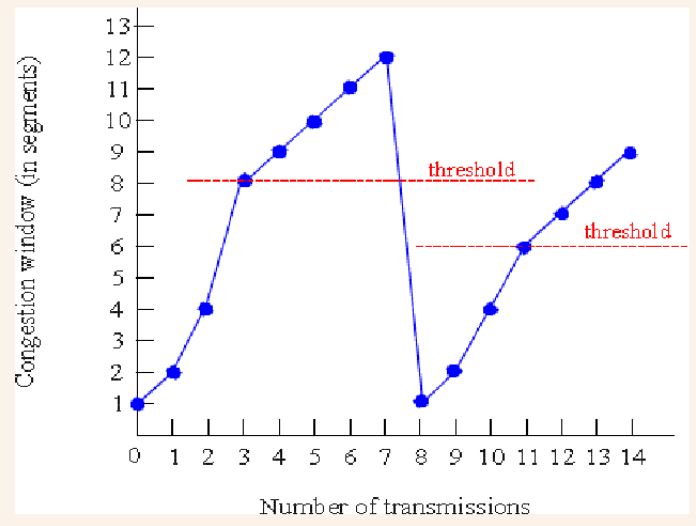

FIGURE 1.1: TCP Tahoe[3].

Il n'y a pas de Fast Recovery, on passe la valeur de ssthresh à la moitié de cwnd courant, cwnd passe à 1 MSS et on retourne en Slow Start.

1.3 RENO

La différence avec Tahoe est qu'il utilise le Fast Recovery. Une fois la réception de trois ACK dupliqués on diminue de moitié la valeur de cwnd, on met le seul de ssthresh à la taille de cwnd, on fait un fast retransmit et on passe en Fast Recovery. Si on a un timeout on repart en Slow Start comme avec Tahoe, avec la fenêtre de congestion à 1 MSS.

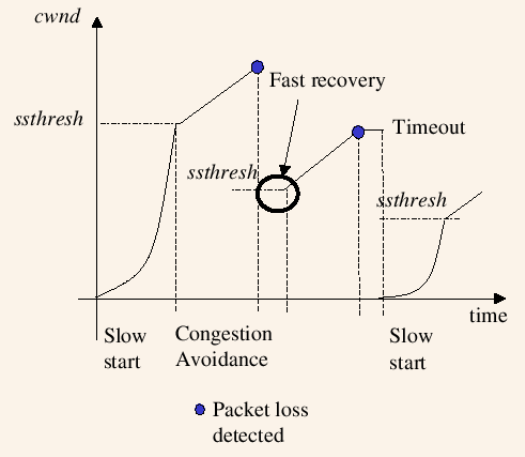

FIGURE 1.2: TCP Reno[3].

Reno permet donc de ne pas repartir en Slow Start (et avec un cwnd à 1) dès qu'il y a congestion. Les débits sont donc plus stables.

1.4 NEW RENO

Cet algorithme s'appelle « NewReno » parce qu'il n'est qu'une modification légère (mais significative) de l'algorithme Reno. En effet, la modification s'opère au niveau de la phase de Fast Recovery : on reste dans ce mode tant que nous n'avons pas reçu les ACK de tous les paquets perdus. Lorsqu'il y a une perte de plusieurs segments d'une même « rafale » envoyée, à la réception d'un acquittement partiel on renvoie le segment perdu suivant, sans sortir du mode Congestion Avoidance, contrairement à Reno qui sort de ce mode dès la réception d'un ACK non dupliqué.

1.5 VEGAS

Plutôt que d'attendre une perte de paquet, Vegas prend en compte le temps de réponse du destinataire (le RTT) afin d'en déduire le ratio auquel envoyer les paquets. En fonction du temps de réponse, on est capable de supposer l'état des buffers des routeurs intermédiaires. Vegas modifie pour cela plusieurs algorithmes vus jusqu'ici (Slow Start, Congestion Avoidance, Fast Retransmit).

Grâce à cette technique Vegas a de meilleurs débits et moins de pertes de paquets que Reno. Cet algorithme permet d'atteindre un partage équitable des ressources. De même, il y a assez peu de pertes avec Vegas puisqu'à l'état stable, le débit correspond de près à la capacité du lien[4].

Puisque Vegas permet de s'adapter plus rapidement aux changements de disponibilité du lien, les performances se dégradent lorsqu'il est utilisé avec d'autres protocoles qui ne prennent pas en compte l'état du lien *avant* une perte de paquet.

Un inconvénient cependant, il nécessite une modification de la pile TCP pour l'émetteur et le récepteur.

1.6 WESTWOOD(+)

Westwood est une réécriture de New Reno côté l'émetteur pour avoir une meilleure gestion des LFN. Il modifie les valeurs de ssthresh et de cwnd en fonction d'estimations qu'il fait sur la bande passante disponible. La première version faisant de mauvaises estimations, elle a été corrigée par la suite, d'où le nom Westwood+. Ces estimations se basent comme pour Vegas sur le temps de réponse du destinataire (RTT), mais ces estimations sont utilisés différemment.

Il possède un mécanisme pour détecter qu'il n'y a pas de congestion (Persistent Non Congestion Detection, PNCD), ce qui lui permet de rapidement utiliser la bande passante disponible. Westwood modifie le Slow Start (il devient moins agressif) en y ajoutant une phase : « Agile Probing », ce qui lui permet (avec les estimations via le RTT) d'avoir une phase d'accroissement du cwnd[2] quasi exponentielle puis de se stabiliser, puis d'accroître, puis de se stabiliser, etc..

L'algorithme gagne beaucoup en efficacité, mais on perd en équité puisqu'en cas de pertes de paquets il ne s'adapte pas brutalement en divisant son cwnd par deux. C'est ce qui fait également qu'il est adapté aux réseaux sans fils, qui ont des pertes aléatoires (et non forcément dues à des congestions).

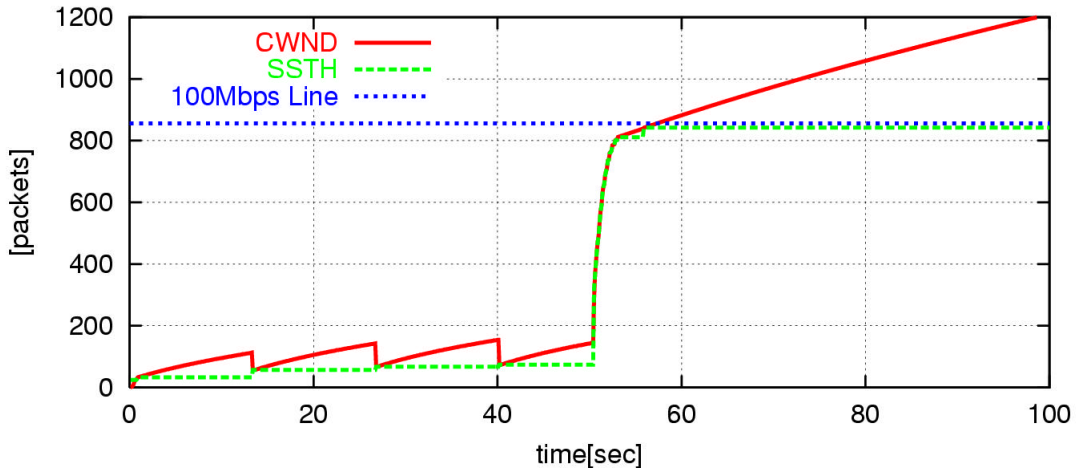

FIGURE 1.3: TCP Westwood - Persistent Non Congestion Detection + Agile Probing[2].

1.7 SACK TCP

Les algorithmes de Slow Start et de Congestion Avoidance sont les mêmes que pour TCP Reno. Selective ACK TCP permet, lors d'une perte, de dire qu'on a reçu les paquets jusqu'à la séquence numéro N, mais aussi de préciser à l'émetteur qu'on a bien reçu certains des paquets suivant. L'émetteur n'a plus à renvoyer des segments déjà reçus. SACK est une option qui se négocie à la connexion. Cette option permet de gagner un peu de performances, lors de pertes de segments, mais il est nécessaire d'avoir les deux parties (émetteur et récepteur) qui l'implémentent.

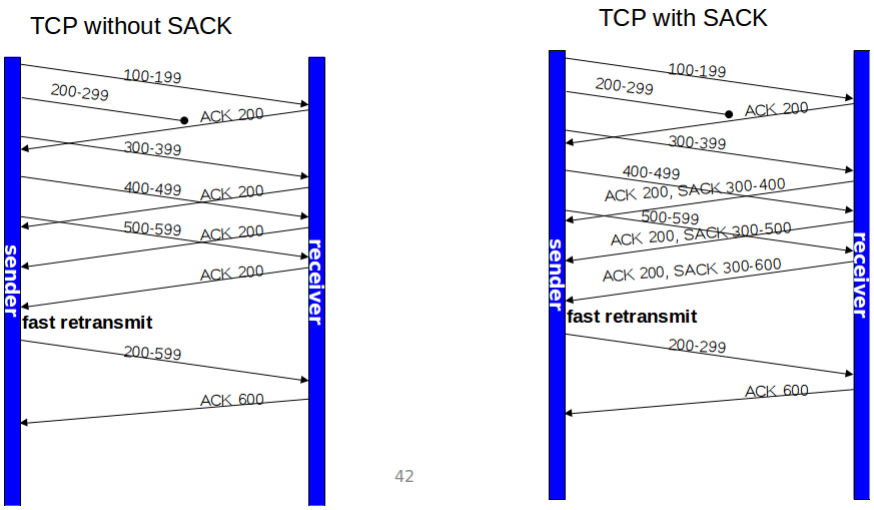

FIGURE 1.4: TCP SACK[1].

2 LES IMPLÉMENTATIONS ACTUELLES

Deux algorithmes d'évitement de congestion sont très présents dans les systèmes d'exploitation courants : CUBIC et CTCP. Avant de voir leurs différences, on va regarder l'algorithme dont CUBIC est hérité, BIC. Le but de ces algorithmes est de garder un débit aussi proche que possible de la bande passante disponible.

2.1 BIC

Les algorithmes précédents ne sont pas adaptés pour des réseaux rapides et avec une importante bande passante (plusieurs gigabits par seconde). Ils laissent une certaine partie de la bande passante non-utilisée. BIC garde un bon débit, une bonne équité même avec les autres algo d'évitement de congestion de TCP.

Une version un peu simplifiée de l'algorithme pourrait être : lorsqu'il y a une perte, BIC réduit cwnd par un certain coefficient. Avant de réduire on va garder en mémoire la valeur de cwnd qui sera notre maximum (Wmax), et la nouvelle valeur sera notre valeur minimum (Wmin). À partir de ces deux valeurs on va rechercher la valeur intermédiaire pour laquelle nous n'avons pas de pertes (recherche dichotomique).

L'algorithme fait attention à ne pas trop augmenter cwnd, aussi si lors de notre recherche on fait des sauts trop grands (défini par un certain seuil), on préfèrera accroître cwnd de manière logarithmique. Une fois la différence amoindrie, on pourra repasser en recherche dichotomique jusqu'à arriver à Wmax. À partir de là on recherche un nouveau maximum, avec une croissance linéaire : on cherche s'il y a un maximum proche. Si après une certaine période on n'a pas de perte, alors on se dit que le maximum est bien plus loin et on augmente de nouveau la taille de la fenêtre de manière exponentielle.

BIC permet une bonne équité, il est stable en maintenant un haut débit et qu'il permet une bonne mise à l'échelle. Malgré ses qualités, il reste trop agressif durant sa phase montante.

Mais surtout il provoque une inégalité avec les flux ayant des RTT longs.

Cet algorithme a été utilisé dans Linux, depuis on lui préfère sa variante plus élaborée : CUBIC.

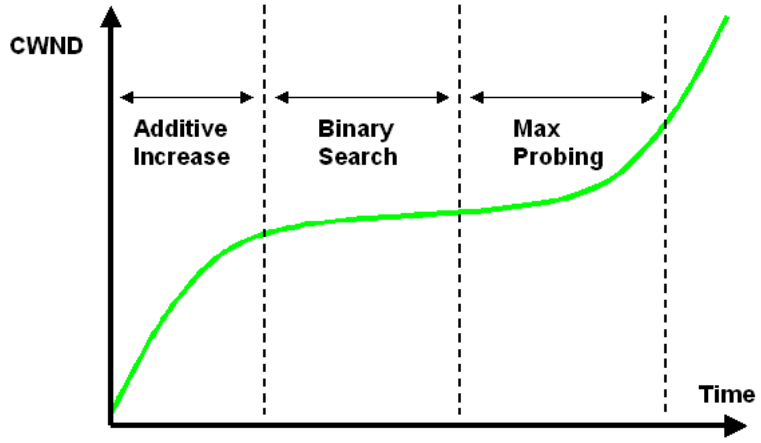

FIGURE 2.1: TCP BIC courbe de fonctionnement (démarrage)

2.2 CUBIC

CUBIC a une phase montante plus douce que celle de BIC, ce qui le rend plus « amical » envers les autres versions de TCP. CUBIC est plus simple car ne possède qu'un seul algorithme pour sa phase montante, et n'utilise pas les acquittements pour augmenter la taille de la fenêtre, on préfèrera parler d'évènement de congestion. C'est ce qui rend CUBIC plus efficace dans les réseaux à bas débit ou avec un RTT court.

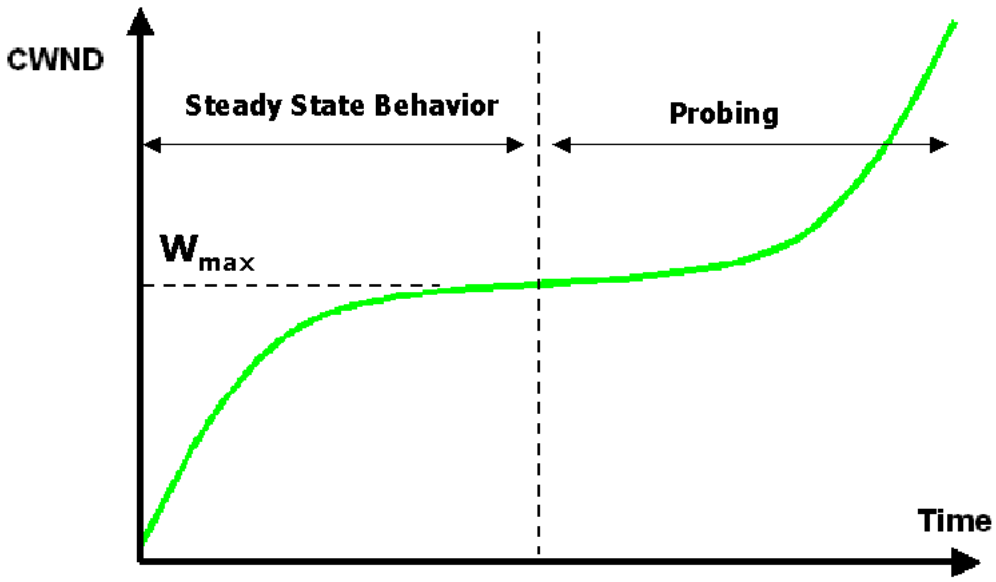

FIGURE 2.2: TCP CUBIC courbe de fonctionnement (démarrage)

2.3 COMPOUND TCP

Une des grandes particularités de CTCP est qu'il maintient deux fenêtres de congestion. La première est la fenêtre qui augmente de façon linéaire mais décroît en cas de perte via un certain coefficient. La seconde fenêtre est liée au temps de réponse du destinataire. On combine donc des méthodes liées à la perte de paquet (comme TCP Reno) et d'adaptation préventive à la congestion (comme TCP Vegas), d'où le nom « Compound » (composé).

La taille de la fenêtre réellement utilisée est la somme de ces deux fenêtres. Si le RTT est bas, alors la fenêtre basée sur le délai augmentera rapidement. Si une perte de segments survient, alors la fenêtre basée sur la perte de paquet diminuera rapidement afin de compenser l'augmentation de la fenêtre basée sur le délai. Avec ce système, on cherchera à garder une valeur constante de la fenêtre d'émission effective, proche de celle estimée.

Le but de ce protocol est de maintenir de bonnes performances sur des réseaux avec un haut débit et avec un grand RTT.

RÉFÉRENCES

[1] Paul D. Amer. Tcp variations : Tahoe, reno, new reno, vegas, sack.

[2] UCLA engineering Anonymous. Tcp westwood with agile probing. 2003.

[3] Eugen Dedu. Contrôle de congestion dans le protocole tcp. 2010.

[4] Larry Peterson Limin Wang Steven Low. Understanding tcp vegas : A duality model. 2000.

[5] wikipedia. Tcp.