Comparaison des principaux algorithmes TCP

Thibaut Ehlinger, *Université de Strasbourg*

22 octobre 2015

TCP (***Transmission Control Protocol***) est une implémentation de la couche transport du modèle OSI. TCP étant omniprésent dans les réseaux actuels, il a donc souvent été étudié et réimplémenté, parfois parce qu'une implémentation présentait des faiblesses qui lui étaient inhérentes, parfois parce qu'aucune implémentation ne répondait à une problématique bien précise du réseau. Dans ce texte, nous étudierons tout d'abord les différentes implémentations historiques de TCP. Celles ayant été utilisées un jour ou l'autre mais qui sont ajourd'hui obsolètes car peu efficaces. Puis nous étudierons certains des algorithmes utilisés aujourd'hui, dans le but de comprendre quels sont les critères déterminants lors du choix de l'algorithme utilisé.

Les premières implémentations de TCP

De TCP Tahoe à TCP Westwood, en passant par TCP Reno, nous allons présenter une série d'algorithmes qui se sont globalement succédés, chacun étant dans la plupart des cas plus efficace que son prédecesseur.

Tahoe

Slow start & congestion avoidance

Cette implémentation est citée pour la première fois en 1988 dans un article de Van Jacobson et Karels. C'est la plus simple et la moins efficace, tout types de problème confondus. Il implémente le **slow start**, la **congestion avoidance**, et le **fast retransmit**. Cela signifie qu'il double la taille de sa CWND à chaque RTT, jusqu'à atteindre le sstresh. A ce moment il passe en congestion avoidance, n'augmentant la taille de sa CWND que de 1 à chaque RTT. En cas de time out, ou d'ACK triplement dupliqué, $sstresh = \frac{CWND_{current}}{2}$, et on repasse en slow start.

Avantages et faiblesses

Tahoe est déjà une réelle avancée, car il permet d'approximer très grossièrement la plus grosse taille de fenêtre d'émission possible, et il reste stable à moins qu'il n'y aie des congestions sur le réseau. Dans ce cas, Tahoe peut s'adapter au problème et diminuer la taille de sa fenêtre d'émission.

Néanmoins, la réinitialisation de la fenêtre d'émission à **1** à la moindre perte de paquet est une sous-estimation beaucoup trop importante. La taille moyenne de CWND est donc bien plus faible que ce qu'elle pourrait être optimalement.

Reno

Reno résoud déjà grandement le problème de l'oscillation perpetuelle Slow Start/CA en ajoutant la notion de **Fast Recovery**.

Fast recovery

Après un fast retransmit, au lieu de passer en Slow Start on applique : $sstresh = \frac{CWND}{2}, CWND = sstresh + 3$. En fait on repasse directement en CA, tout en augmentant la taille de CWND de 3, en référence aux trois segments qui n'ont pas été reçus à cause des ACK dupliqués. En cas de time out néanmoins, on repasse en Slow Start. Cette stratégie permet en cas de perte de ne pas baisser drastiquement la taille de CWND, à moins d'un time out. Un time out est en fait plus grave qu'une simple perte de paquet. Il témoigne probablement d'une modification topologique du réseau ou bien d'une congestion importante, alors qu'une simple perte de paquet peut-être due à des évènements plus ponctuels, tels qu'un déséquencement ou une perte.

Faiblesses

TCP Reno réagit bien au pertes de paquets quand elles se limitent à une par rafale. Quand par contre il y en a plusieurs par rafale, Reno est presque aussi inefficace que Tahoe, car il ne peut détecter qu'une seule perte de paquet à la fois.

De plus, il se peut qu'au sein d'une même rafale, l'émetteur aie le temps de passer en Fast Recovery, puis à nouveau en Congestion Avoidance, **deux fois**. Ce qui veut dire que si la fenêtre est trop grande et

les pertes trop espacées au sein de cette fenêtre, la CWND peut-être divisée par 4.

New Reno

New Reno peut détecter les pertes de paquets multiples et est, de fait, plus performant que Reno. En effet, New Reno garde une trace de tous les segments envoyés dans une rafale. Lorsqu'il détecte une perte, il **reste** en fast recovery tant que tous les segments de cette rafale n'ont pas été acquittés par le destinataire. New Reno est mieux que Reno en tout point.

Problèmes

Puisque les pertes ne sont détectées que par les duplications d'acquittements, il faut tout de même attendre un timeout entier pour détecter toutes les pertes.

SACK

ACK sélectifs

SACK permet de nommer les segments non-reçus, contrairement à Reno qui se contente de dupliquer les acquittement. Cela permet de récupérer plus vite les paquets perdus.

Variable *pipe*

En Fast Recovery, SACK initialise une variable *pipe*. C'est une estimation du nombre de paquets qui sont encore dans le réseau. Si SACK reçoit une duplication d'ACK, il incrémente pipe. Il le décrémente pour toute nouvelle transmission ou nouvel ACK. Quand $pipe < CWD$, il renvoie tous les segements qui n'ont pas été acquittés. Cela permet de renvoyer plusieurs segments perdus en moins d'un RTT.

Problèmes de SACK

SACK est nettement plus efficace que New Reno, mais il présente un défaut majeur : il doit être implémenté par l'émetteur **et** le récepteur, sinon il ne marchera pas.

Vegas

Mécanisme de retransmission

D'après les travaux de Lawrence Brakmo et Larry L. Peterson, deux chercheurs de l'université d'Arizona et concepteurs de Vegas, Reno et New Reno sont très peu performants en ce qui concernt les **time out**. En effet, ils ont mesuré que pour un RTT moyen de 500ms, il fallait approximativement **1100ms** à New Reno pour détecter un time out.[1] Ce qui correspond effectivement à plus de deux RTT. Vegas mesure le temps écoulé entre l'envoi d'un segment et la réception de son acquittement. Et mesure ainsi dynamiquement les RTT. Cela permet non seulement d'estimer le time out plus précisément et de perdre moins de temps en cas de réinitialisation déclenchée par un time out, mais non seulement de détecter les pertes et les déséquencements préventivement :

En cas d'ACK dupliqué , on compare le nouveau RTT avec le RTT mesuré précédemment. Si elle est supérieure à Time Out value, on renvoie directement le segment suivant, sans attendre trois acquittements dupliqués.

Après un ACK dupliqué on fait de même. En effet, il est moins couteux de renvoyer un segment que de perdre beaucoup de temps pour découvrir qu'il a été perdu.

De plus, Vegas ne diminue la taille de sa CWND que si le dernier ACK renvoyé a été renvoyé **après** le dernier changement de taille de fenêtre. En effet on peut supposer que deux pertes peuvent avoir été provoquées par la même congestion et qu'il ne sert à rien d'y réagir deux fois.

Congestion avoidance

Pour l'évitement de congestions, Vegas fait une estimation du débit *attendu* et le compare avec le débit *estimé*. Ensuite on fixe des seuils $\alpha < \beta$. Si la différence des débits est inférieure à α alors on augmente CWND linéairement, si elle est supérieure à β, on la diminue linéairement.

Congestion avoidance

En Slow Start, Vegas n'augmente exponentiellement que tout les RTT, donc moins fréquemment que ses comparses. De plus, on fixe expérimentalement un seuil γ. Si la différence entre le RTT *estimé* et le *attendu* est inférieure à ce γ, on repasse en CA.

[1] *In Reno, round trip time (RTT) and variance estimates are computed using a coarse-grained timer (around 500 ms), [...] we found that for losses that resulted in a timeout—usually due to two or more dropped segments in a RTT , the exact figure depends on the number of segments in 4 transit—it took Reno an average of 1100ms.*

Aggressivité

La courbe de Vegas est donc très ronde puisque l'algorithme tente d'anticiper les problèmes de congestion. Ces variations sont beaucoup plus lentes que Reno, par exemple. On dit que Vegas est **peu aggressif**. Contrairement à Reno ou à Tahoe, il ne tente pas de prendre le plus de bande passante que possible. Il en prend plus si il sent que le réseau lui permet. Si Reno intervient sur le même réseau que Vegas, alors les performances de Vegas vont diminuer considérablement, étant donné que Vegas diminuera de sa propre initiative son débit d'émission en amont de la congestion provoquée par Reno. Vegas cède en quelque sorte la priorité aux autres algorithmes cités précedemment.

Westwood

Westwood a été conçu pour les réseaux connaissant un taux de perte de paquet important malgré une absence de congestion, tels que les réseaux sans fil. Comme ces pertes sont inhérentes au réseau, il ne sert à rien de compter directement les paquets perdus, car le même taux de paquet sera perdu quelque soit la taille de CWND. Du coup, Westwood fait une estimation du débit du réseau en divisant la taille d'un paquet envoyé par l'intervalle de temps entre les deux derniers acquittements. En cas de duplication d'ACK ou de time out, on se base sur cette estimation pour fixer sstresh : $sstresh = \frac{BWE \times RTT_{min}}{segsize}$. Westwood se base donc sur le débit moyen du réseau au lieu de compter les pertes.

Aggessivité

Westwood est "amical" (*friendly*). D'après les fondateurs de Westwood, lorsqu'il est en concurrence avec Reno, il partage la connexion très équitablement. De plus, TCP Westwood est considérablement plus performant que Reno lorsqu'il y a des pertes de 1% dans le réseau, ce qui est un taux crédible dans des réseaux non-filaires.

Les implémentations modernes de TCP

BIC (*Binary Increase Congestion Control*

BIC est une implémentation TCP optimisée pour les *Long fat network*, c'est à dire les réseaux à grande latence et à à grand débit, tels que les réseaux satellitaires. Dans ces réseaux, il faut trouver la CWND optimale en provoquant le moins de perte de paquet possible. Pour ce faire, BIC fait une recherche binaire, recherche réputée pour sa complexitée logarithmique.

Protocole

Pour procéder à sa recherche binaire, BIC enregistre W_{max}, la taille maximum de CWND avant le dernire Fast Recovery, et W_{min}, la taille de la fenêtre juste **avant** le dernier fast recovery. La moyenne de W_{max} et W_{min}, W_{temp} est utilisée comme CWND. On augmente CWND linéairement jusqu'à W_{max}. S'il y a perte de paquet, W_{temp} devient le dernier W_{max}, sinon il devient le dernier W_{min}. La taille de la fenêtre $\{W_{max}\ W_{min}\}$ est divisée par deux à chaque itération. Afin de ne pas imposer de changement de débit trop violent au réseau, on plafonne l'augmentation de la CWND arbitrairement avec un indice S_{max}.

Quand CWND dépasse W_{max}, il faut trouver une nouvelle taille de fenêtre maximale. On entre alors en "Max probing" : on augmente d'abord CWND linéairement, de S_{max}, en supposant qu'il n'est pas loin, puis au bout d'un certain temps, s'il n'y a pas de perte, on suppose que W_{max} est très élevé. Dans ce cas on augmente CWND de $N \times S_{max}$. N étant le nombre d'itérations de la phase Incremental Addition du max probing.

CUBIC

BIC est certes avantageux pour les *LFN*, mais il est très aggressif, dans le sens ou il augmente très rapidement et très fortement sa CWND. C'est normal, c'est le but de BIC. Pour des réseaux à faible débit ou faible RTT, cela peut-être néanmoins très contraignant.

Protocole

Cubic se veut lui aussi souple et stable, mais beaucoup moins aggressif. La taille de CWND est déterminée par une fonction cubique :

$W_{CUBIC} = C(t-K)^3 - W_{max}$, avec C déterminé empiriquement, t le temps écoulé depuis depuis la dernière réduction de fenêtre, W_{max} la taille de la fenêtre juste avant la dernière réduction de fenêtre et

$K = \sqrt[3]{W_{max} \times \frac{\beta}{C}}$, β étant le facteur de réduction de la CWND lors de la dernière réduction. Derrière cette fonction complexe se cachent plusieurs fait :

La fonction est **cubique**, donc elle croît rapidement puis se stabilise autour de la W_{max}, pour recroître rapidement.

Elle dépend de l'intervalle de temps écoulé depuis la dernière réduction ce qui signifie que **tous** les processus Cubic TCP d'un même réseau croîtront à la même vitesse, garantissant l'équité de l'algorithme envers différents agents TCP.

Si le RTT est faible, la fonction croît lentement. Ce qui résoud le problème d'aggressivité posé posé par BIC, car CUBIC est "amical" envers toutes les implémentations de TCP.

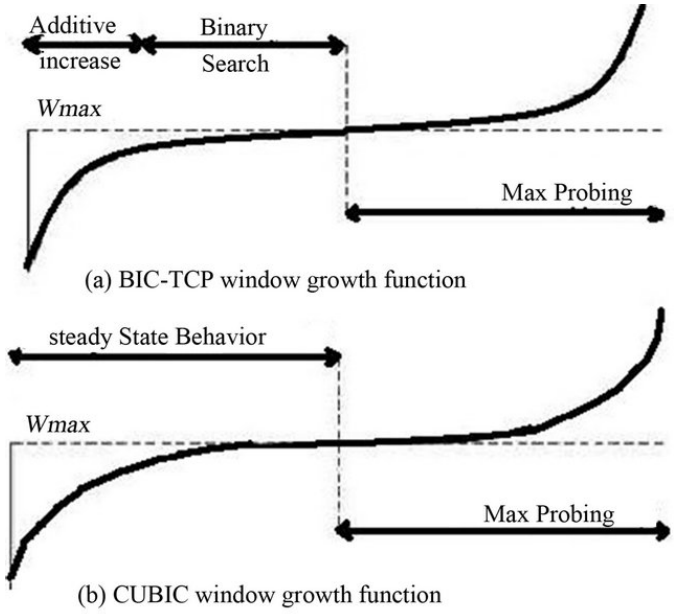

Figure 1: *Sur ce schéma, on voit que CUBIC est plus rond que BIC autour de W_{Max}*

Compound

Compound (qui signifie en anglais *compsé*) tient son nom du fait qu'il est un hybride entre deux algorithmes TCP. L'idée derrière ce principe est que si le lien est sous-utilisé, alors il faut augmenter la taille de CWND le plus rapidement possible, mais que s'il est pleinement utilisé, alors il faut limiter l'aggressivité de l'algorithme, afin de garantir le *TCP fairness*. La CWND est la somme de deux fenêtre : une *delay-window* et une *AIMD-Window*. Bien que AIMD (*additive-increase/multiplicative-decrease*) ne soit pas traité dans ce texte, on peut noter que cet algorithme augmente linéairement et est divisé par un facteur fixé arbitrairement en cas de congestion. En cas de perte de paquet, et qu'on entre en congestion avoidance, la *delay-window* prend de plus en plus d'ampleur selon un algorithme très similaire à TCP Vegas, tandis que la *AIMD-Window* a de moins en moins d'importance puisqu'elle augmente désormais très lentement.

Y a-t-il un meilleur TCP?

Critères

Il y a trois critères pour comparer l'efficacité des algorithmes :

Son équité : si plusieurs agents de cet algorithme tournent sur un même réseau, vont-ils se partager la bande passante équitablement?

Sa compatibilité : cet algorithme va-t-il être trop aggressif ou trop pacifique face à d'autres algorithmes TCP sur le réseau?

Son efficacité : cet algorithme monopolise-t-il en moyenne une proportion suffisante du lien?

Bilan

De nombreuses simulations ont montré que les premiers algorithmes de TCP sont désormais obsolètes. Vegas est incompatible avec Reno, Tahoe est trop oscillant, Reno/New Reno mettent trop de temps à détecter les pertes multiples, SACK est difficile à mettre en place et Westwood présente certains défauts qui ont été corrigés dans Westwood+.

Pour les implémentations modernes de TCP, le choix de l'algorithme dépend de la topologie du réseau. En effet, BIC est conçu pour les *Long Fat Network*, tandis que CUBIC réagit mieux sur les réseaux à faible latence. Il est donc conseillé d'expérimenter différentes implémentations sur le réseau cible et de regarder laquelle est la plus efficace, en fonction du critère que l'on priorise. L'équité par exemple paraît très importante sur un réseau très peuplé, la compatibilité le paraît encore plus si l'on n'est pas maître des autres agents utilisant le réseau. Enfin si c'est un réseau peu peuplé, alors on peut se baser sur l'efficacité de l'algorithme lorsqu'on l-e sélectionne.

References

[1] Lawrence S. Brakmo, Sean W. O'Malley, Larry L. Peterson, *TCP Vegas : New Techniques for Congestion Detection and Avoidance*, SIGCOMM Comput. Commun. Rev., 1994

[2] Luigi A. Grieco, Saverio Mascolo , *Performance Evaluation and Comparison of Westwood+, New Reno, and Vegas TCP Congestion Control* , SIGCOMM Comput. Commun. Rev., 2004

[3] Claudio Casetti, Mario Gerla, Saverio Mascolo, M. Y. Sanadidi, Ren Wang *TCP Westwood: End-to-End Congestion Control for Wired/Wireless Networks*, Wirel. Netw., 2002

[4] Lisong Xu, Khaled Harfoush, Injong Rhee *Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks*, INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, 2004

[5] Injong Rhee, Lisong Xu, *CUBIC: A New TCP-Friendly High-Speed TCP Variant*, SIGOPS Oper. Syst. Rev., 2008

[6] Kun Tan, Jingmin Song, Qian Zhang, Murari Sridharan *A Compound TCP Approach for High-speed and Long Distance Networks* , INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings , 2006

[7] *BIC and CUBIC* http://research.csc.ncsu.edu/netsrv/?q=content/bic-and-cubic