RFC 8985 : The RACK-TLP Loss Detection Algorithm for TCP

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 23 février 2021

Date de publication du RFC : Février 2021

https://www.bortzmeyer.org/8985.html

Ce RFC normalise un nouvel algorithme pour la détection de paquets perdus dans TCP. Dans des cas comme celui où l'application n'a rien à envoyer (fin des données, ou attente d'une requête), RACK-TLP détecte les pertes plus rapidement, en utilisant des mesures fines du RTT, et en sollicitant des accusés de réception dans de nouveaux cas.

Pour comprendre cet algorithme et son intérêt, il faut revoir à quoi sert un protocole de transport comme TCP. Les données sont transmises sous forme de paquets (au format IP, dans le cas de l'Internet), paquets qui peuvent être perdus en route, par exemple si les files d'attente d'un routeur sont pleines. Le premier travail d'un protocole de transport est de réémettre les paquets perdus, de façon à ce que les applications reçoivent bien toutes les données (et dans l'ordre car, en prime, les paquets peuvent arriver dans le désordre, ce qu'on nomme un réordonnancement, qui peut être provoqué, par exemple, lorsque deux chemins sont utilisés pour les paquets). Un protocole de transport a donc besoin de détecter les pertes. Ce n'est pas si évident que cela. La méthode « on note quand on a émis un paquet et, si on n'a pas reçu d'accusé de réception au bout de N millisecondes, on le réémet » fonctionne, mais elle serait très inefficace. Notamment, le choix de N est difficile : trop court, et on déclarerait à tort des paquets comme perdus, réémettant pour rien, trop long et et ne détecterait la perte que trop tard, diminuant la capacité <https://www.bortzmeyer.org/capacite.html> effective.

Le RFC original sur TCP, le RFC 793 [1], expliquait déjà que N devait être calculé dynamiquement, en tenant compte du RTT attendu. En effet, si l'accusé de réception n'est pas arrivé au bout d'une durée égale au RTT, attendre davantage ne sert à rien. (Bien sûr, c'est plus compliqué que cela : le RTT peut être difficile à mesurer lorsque des paquets sont perdus, il varie dans le temps, puisque le réseau a pu changer, et il peut y avoir des temps de traitement qui s'ajoutent au RTT, il faut donc garder un peu de

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc793.txt

marge.) Armé de cette mesure dynamique du RTT, TCP peut calculer un RTO ("*Retransmission TimeOut*", RFC 793, section 3.7) qui donne le temps d'attente. Le concept de RTO a été ensuite détaillé dans le RFC 6298 qui imposait un minimum d'une seconde (ce qui est beaucoup, par exemple à l'intérieur d'un même centre de données).

Mais d'autres difficultés surgissent ensuite. Par exemple, les accusés de réception de TCP indiquent le dernier octet reçu. Si on envoie trois paquets (plus rigoureusement trois segments, le terme utilisé par TCP) de cent octets, et qu'on reçoit un accusé de réception pour le centième octet indique que le premier paquet est arrivé mais ne dit rien du troisième. Peut-être est-il arrivé et que le second est perdu. Dans ce cas, la réémission du troisième paquet serait du gaspillage. Les accusés de réception cumulatifs de TCP ne permettent pas au récepteur de dire qu'il a reçu les premier et troisième paquets. Les SACK ("*Selective Acknowledgments*") du RFC 2018 ont résolu ce problème. Dans l'exemple ci-dessus, SACK aurait permis au récepteur de dire « j'ai reçu les octets 0 - 100 et 200 - 300 [le second nombre de chaque bloc indique l'octet suivant le bloc] ».

Ensuite, on peut optimiser l'opération en n'attendant pas le RTO, dont on a vu qu'il était souvent trop long. L'idée de base est que, si le récepteur reçoit des données qui ne sont pas dans l'ordre attendu (le troisième paquet, alors qu'on n'a toujours pas vu le second), on renvoie un accusé de réception pour les données déjà reçues et déjà confirmées. Lorsque l'émetteur reçoit un accusé de réception qu'il a déjà vu (en fait, plusieurs), il comprend que des données manquent et il peut réémettre tout de suite, même si le RTO n'est pas écoulé. Ce mécanisme d'accusés de réception dupliqués (DUPACK, pour "*DUPlicate ACKnowledgment*") a été décrit dans le RFC 5681, puis le RFC 6675. Une de ses faiblesses, que corrige notre RFC, est qu'il ne peut pas être utilisé à la fin d'une transmission, puisqu'il n'y a plus de données qui arrivent, empêchant le récepteur de détecter les pertes ou réordonnancements, et obligeant à attendre le RTO.

Et enfin, il faut se rappeler que le protocole de transport a une autre importante responsabilité : s'il doit s'assurer que toutes les données arrivent, et donc réémettre les paquets manquants, il doit aussi le faire en évitant la congestion (RFC 5681). Si des paquets se perdent, c'est peut-être que le réseau est saturé et réémettre trop brutalement pourrait encore aggraver la situation. Ceci dit, l'algorithme RACK-TLP décrit dans notre RFC ne traite que de la détection de pertes, les méthodes précédentes pour le contrôle de congestion restent valables. (C'est une des nouveautés de RACK-TLP : il sépare la détection de pertes du contrôle de congestion, pour lequel il existe plusieurs algorithmes comme le NewReno du RFC 6582 ou le CUBIC du RFC 8312. Ce découplage permet de faire évoluer séparement les algorithmes, et rend donc le système plus souple.)

Voici, désolé pour cette introduction un peu longue, on peut maintenant passer au sujet principal de ce RFC, RACK-TLP. Cet algorithme, ou plutôt ces deux algorithmes, vont être plus efficaces (détecter les pertes plus vite) que DUPACK, notamment à la fin d'une transmission, ou lorsqu'il y a des pertes des paquets retransmis, ou encore des réordonnancements fréquents des paquets (section 2.2 de notre RFC). RACK ("*Recent ACKnowledgment*") va utiliser le RTT des accusés de réception pour détecter certaines pertes, alors que TLP ("*Tail Loss Probe*") va émettre des paquets de données pour provoquer l'envoi d'accusés de réception par le récepteur.

La section 3 du RFC donne une vision générale des deux algorithmes. Commençons par RACK ("*Recent ACKnowledgment*"). Son but va être de détecter plus rapidement les pertes lorsque un paquet arrive au récepteur dans le désordre (paquet des octets 200 à 300 alors que le précédent était de 0 à 100, par exemple). Il va utiliser SACK (RFC 2018) pour cela, et RACK dépend donc du bon fonctionnement de SACK (section 4) et en outre, au lieu de ne regarder que les numéros de séquence contenus dans les ACK, comme le faisait DUPACK, il va également regarder le temps écoulé depuis l'émission d'un paquet. Les détails figurent en section 6 du RFC.

Quant à TLP ("*Tail Loss Probe*"), son rôle est de titiller le récepteur pour que celui-ci émette des accusés de récéption, même s'il n'en voyait pas la nécessité. Par exemple, si on arrive à la fin d'une session, l'émetteur a fini d'envoyer ses paquets, mais pas encore reçu les accusés de réception. Aucun envoi ne déclenchera donc DUPACK. Le principe de TLP est donc de solliciter un accusé de réception pour voir si on obtient un duplicata. RACK pourra alors utiliser cet accusé dupliqué. (Les deux algorithmes sont forcément utilisés ensemble, on n'utilise pas TLP sans RACK.) Les détails de TLP sont dans la section 7.

La résolution doit être meilleure qu'un quart du RTT. À l'intérieur d'un centre de données, cela implique de pouvoir compter les microsecondes, et, sur l'Internet public, les millisecondes.

La section 9 du RFC discute des avantages et inconvénients de RACK-TLP. Le gros avantage est que tout segment de données envoyé, **même si c'est une retransmission**, peut permettre de détecter des pertes. C'est surtout intéressant lorsqu'on est à la fin des données (et il n'y a donc plus rien à transmettre) ou lorsque des retransmissions sont perdues. Outre le cas de la fin du flot de données, les propriétés de RACK-TLP sont également utiles lorsque l'application garde la connexion ouverte mais n'a temporairement rien à transmettre (cas de sessions interactives, ou de protocoles requête/réponse comme EPP, où il ne se passe pas forcément quelque chose en permanence).

Mais comme la perfection n'existe pas en ce bas monde, RACK-TLP a aussi des inconvénients. Le principal est qu'il oblige à garder davantage d'état puisqu'il faut mémoriser l'heure de départ de chaque segment (contrairement à ce que proposait le RFC 6675), et avec une bonne résolution (un quart du RTT, demande le RFC). Une telle résolution n'est pas facile à obtenir dans un centre de données où les RTT sont très inférieurs à la milliseconde. Ceci dit, certaines mises en œuvre de TCP font déjà cela, même si, en théorie, elles pourraient se contenter de moins.

RACK-TLP coexiste avec d'autres algorithmes de détection de pertes, comme le classique RTO (RFC 6298) mais aussi avec ceux des RFC 7765, RFC 5682 et RFC 3522. De même, il coexiste avec divers algorithmes de contrôle de la congestion (RFC 5681, RFC 6937) puisqu'il se contente de détecter les pertes, il ne décide pas de comment on évite la congestion. En gros, RACK-TLP dit « ce segment est perdu » et la décision de réémission de celui-ci passera par l'algorithme de contrôle de la congestion, qui verra quand envoyer ces données à nouveau.

RACK-TLP est explicitement conçu pour TCP, mais il pourra aussi être utilisé dans le futur avec d'autres protocoles de transport, comme le futur QUIC.

RACK-TLP ne date pas d'aujourd'hui et il est déjà mis en œuvre dans Linux, FreeBSD et dans des systèmes d'exploitation moins connus comme Windows.