RFC 8257 : Data Center TCP (DCTCP): TCP Congestion Control for Data Centers

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 19 octobre 2017

Date de publication du RFC : Octobre 2017

https://www.bortzmeyer.org/8257.html

DCTCP ("*Datacenter TCP*"), décrit dans ce RFC (qui n'est pas une norme, attention), est une variante de TCP avec un contrôle de congestion moins prudent. Elle est conçue pour le cas particulier des centres de données et ne doit **pas** être utilisée sur l'Internet public.

DCTCP repose sur la technique ECN du RFC 3168[1]. Normalement, cette technique permet de signaler la congestion plus tôt qu'avec la méthode traditionnelle d'attendre les pertes de paquets. L'ECN est binaire : il y a eu de la congestion ou il n'y en a pas eu. DCTCP va plus loin et utilise ECN pour estimer le nombre d'octets qui ont rencontré de la congestion. DCTCP va ensuite refermer sa fenêtre (le nombre d'octets qu'on peut envoyer avant d'avoir reçu des accusés de réception) plus lentement que le TCP classique (et c'est pour cela que la concurrence entre eux est inégale, et que DCTCP ne doit pas être activé dans l'Internet, mais seulement dans des environnements fermés).

Quelles sont les particularités du réseau dans un centre de données ? Il faut beaucoup de commutateurs pour relier ces machines. Il est tentant d'utiliser pour cela des commutateurs bon marché. Mais ceux-ci ont des tampons de taille limitée, et le risque de congestion est donc plus élevé. Dans un centre de données, les flots sont de deux types : des courts et des longs. Les courts veulent en général une faible latence <https://www.bortzmeyer.org/latence.html> et les longs sont souvent davantage intéressés par une forte capacité <https://www.bortzmeyer.org/capacite.html>. Enfin, le trafic est souvent très synchronisé. Si on fait du "*MapReduce*", tous les serveurs vont voir de l'activité réseau en même temps (quand le travail est réparti, et quand il se termine).

Le cahier des charges des commutateurs est donc plein de contradictions :
— Tampon de petite taille, pour ne pas augmenter la latence (attention au "*bufferbloat*"),

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc3168.txt

— Tampon de grande taille pour être sûr de pouvoir toujours utiliser les liens de sortie au maximum,
— Tampon de grande taille pour pouvoir encaisser les brusques variations de trafic, lorsqu'un flot bavard commence (par exemple une distribution d'un nouveau travail "*MapReduce*").

Avec le TCP traditionnel (pré-ECN), l'indicateur de congestion est la perte de paquets, détectée par l'absence d'accusé de réception. (Voir le RFC 5681 pour une bonne synthèse sur le contrôle de congestion dans TCP.) Attendre la perte de paquets pour ralentir n'est pas très efficace : pour un flot court qui rencontre de la congestion au début, la majorité des paquets aura été jetée avant que TCP ne puisse ralentir. D'où l'invention d'ECN (RFC 3168). ECN permet de réagir **avant** qu'on perde des paquets. Mais, comme expliqué plus haut, il est binaire : il détecte la congestion, pas son importance. Il va donc souvent mener TCP à refermer trop énergiquement la fenêtre d'envoi.

La section 3 du RFC présente les algorithmes à utiliser. Les commutateurs/routeurs détectent la congestion et la signalent via ECN (RFC 3168). Les récepteurs des données renvoient l'ECN à l'émetteur et celui-ci réduit sa fenêtre de congestion (`cwnd` pour "*Congestion WiNDow*", cf. RFC 5681, section 2). Tout ceci est le fonctionnement classique d'ECN. C'est surtout dans la dernière étape, le calcul de la réduction de la fenêtre, que DCTCP apporte des nouveautés. Mais, avant, quelques détails sur les deux premières étapes.

D'abord, la décision des commutateurs et·ou routeurs de considérer qu'il y a congestion. Fondamentalement, c'est une décision locale, qui n'est pas standardisée. En général, on décide qu'il y a congestion dès que le temps de séjour des paquets dans les tampons du commutateur/routeur augmente « trop ». On n'attend donc pas que les files d'attente soient pleines (si elles sont grandes - "*bufferbloat*" - la latence va augmenter sérieusement bien avant qu'elles ne soient pleines). Une fois que l'engin décide qu'il y a congestion, il marque les paquets avec ECN (bit CE - "*Congestion Experienced*", cf. RFC 3168).

Le récepteur du paquet va alors se dire « ouh là, ce paquet a rencontré de la congestion sur son trajet, il faut que je prévienne l'émetteur de se calmer » et il va mettre le bit ECE dans ses accusés de réception. Ça, c'est l'ECN normal. Mais pour DCTCP, il faut davantage de détails, puisqu'on veut savoir précisément quels octets ont rencontré de la congestion. Une possibilité serait d'envoyer un accusé de réception à chaque segment (paquet TCP), avec le bit ECE si ce segment a rencontré de la congestion. Mais cela empêcherait d'utiliser des optimisations très utiles de TCP, comme les accusés de réception retardés (on attend un peu de voir si un autre segment arrive, pour pouvoir accuser réception des deux avec un seul paquet). À la place, DCTCP utilise une nouvelle variable booléenne locale chez le récepteur qui stocke l'état CE du précédent segment. On envoie un accusé de réception dès que cette variable change d'état. Ainsi, l'accusé de réception des octets M à N indique, selon qu'il a le bit ECE ou pas, que tous ces octets ont eu ou n'ont pas eu de congestion.

Et chez l'émetteur qui reçoit ces nouvelles notifications de congestion plus subtiles ? Il va s'en servir pour déterminer quel pourcentage des octets qu'il a envoyé ont rencontré de la congestion. Les détails du calcul (dont une partie est laissé à l'implémenteur, cf. section 4.2) figurent en section 3.3. Le résultat est stocké dans une nouvelle variable locale, `DCTCP.Alpha`.

Une fois ces calculs faits et cette variable disponible, lorsque la congestion apparait, au lieu de diviser brusquement sa fenêtre de congestion, l'émetteur la fermera plus doucement, par la formule `cwnd = cwnd * (1 - DCTCP.Alpha / 2)` (où `cwnd` est la taille de la fenêtre de congestion ; avec l'ancien algorithme, tout se passait comme si tous les octets avaient subi la congestion, donc `DCTCP.Alpha` = 1).

La formule ci-dessus était pour la cas où la congestion était signalée par ECN. Si elle était signalée par une perte de paquets, DCTCP se conduit comme le TCP traditionnel, divisant sa fenêtre par deux. De même, une fois la congestion passée, "*Datacenter TCP*" agrandit sa fenêtre exactement comme un TCP normal.

Voilà, l'algorithme est là, il n'y a plus qu'à le mettre en œuvre. Cela mène à quelques points subtils, que traite la section 4. Par exemple, on a dit que DCTCP, plus agressif qu'un TCP habituel, ne doit pas rentrer en concurrence avec lui (car il gagnerait toujours). Une implémentation de DCTCP doit donc savoir quand activer le nouvel algorithme et quand garder le comportement conservateur traditionnel. (Cela ne peut pas être automatique, puisque TCP ne fournit pas de moyen de négocier l'algorithme de gestion de la congestion avec son pair.) On pense à une variable globale (configurée avec sysctl sur Unix) mais cela ne suffit pas : la même machine dans le centre de données peut avoir besoin de communiquer avec d'autres machines du centre, en utilisant DCTCP, et avec l'extérieur, où il ne faut pas l'utiliser. Il faut donc utiliser des configurations du genre « DCTCP activé pour les machines dans le même /48 que moi ».

Une solution plus rigolote mais un peu risquée, serait d'activer DCTCP dès que la mesure du RTT indique une valeur inférieure à N millisecondes, où N est assez bas pour qu'on soit sûr que seules les machines de la même tribu soient concernées.

Après le programmeur en section 4, l'administrateur réseaux en section 5. Comment déployer proprement DCTCP ? Comme on a vu que les flots TCP traditionnels et DCTCP coexistaient mal, la section 5 recommande de les séparer. Par exemple, l'article « *"Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter"* <https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/judd> » décrit un déploiement où le DSCP (RFC 2474) d'IPv4 est utilisé pour distinguer les deux TCP, ce qui permet d'appliquer de l'AQM (RFC 7567) à DCTCP et des méthodes plus traditionnelles (laisser tomber le dernier paquet en cas de congestion) au TCP habituel. (Il faut aussi penser au trafic non-TCP, ICMP, par exemple, quand on configure ses commutateurs/routeurs.)

Aujourd'hui, DCTCP est déjà largement déployé et ce RFC ne fait que prendre acte de ce déploiement On trouve DCTCP dans Linux (cf. ce commit de 2014 <https://git.kernel.org/cgit/linux/kernel/git/davem/net-next.git/commit/?id=e3118e8359bb7c59555aca60c725106e6d78c5ce>, notez les mesures de performance qui accompagnent sa description), dans FreeBSD (ce commit <https://github.com/freebsd/freebsd/commit/8ad879445281027858a7fa706d13e458095b595f>, et cette description de l'implémentation <https://www.bsdcan.org/2015/schedule/events/559.en.html>), et sur Windows (cette fois, on ne peut pas voir le source mais il y a une documentation <https://technet.microsoft.com/en-us/library/hh997028(v=ws.11).aspx>). Sur Linux, on peut voir la liste des algorithmes de gestion de la congestion qui ont été compilés dans ce noyau :

```
% sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = cubic reno
```

Si DCTCP manque, c'est peut-être parce qu'il faut charger le module :

```
% modprobe tcp_dctcp
% sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = cubic reno dctcp
```

Si DCTCP se trouve dans la liste, on peut l'activer (c'est une activation globale, par défaut) :

```
% sysctl -w net.ipv4.tcp_congestion_control=dctcp
```

Pour le faire uniquement vers certaines destinations (par exemple à l'intérieur du centre de données) :

```
% ip route add 192.168.0.0/16 congctl dctcp
```

Le choix des algorithmes de gestion de la congestion peut également être fait par chaque application (`setsockopt(ns, IPPROTO_TCP, TCP_CONGESTION, ...)`).

Enfin, la section 6 du RFC rassemble quelques problèmes non résolus avec DCTCP :

— Si les estimations de congestion sont fausses, les calculs de DCTCP seront faux. C'est particulièrement un problème en cas de perte de paquets, problème peu étudié pour l'instant.

— Comme indiqué plus haut, DCTCP n'a aucun mécanisme pour négocier dynamiquement son utilisation. Il ne peut donc pas coexister avec le TCP traditionnel mais, pire, il ne peut pas non plus partager gentiment le réseau avec un futur mécanisme qui, lui aussi, « enrichirait » ECN. (Cf. la thèse de Midori Kato <https://eggert.org/students/kato-thesis.pdf>.)

Enfin, la section 7 du RFC, portant sur la sécurité, note que DCTCP hérite des faiblesses de sécurité d'ECN (les bits ECN dans les en-têtes IP et TCP peuvent être modifiés par un attaquant actif) mais que c'est moins grave pour DCTCP, qui ne tourne que dans des environnements fermés.

Si vous aimez lire, l'article original décrivant DCTCP en 2010 est celui de Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., et M. Sridharan, « "*Data Center TCP (DCTCP)*" <http://dl.acm.org/citation.cfm?doid=1851182.1851192> ». Le dinosaure ACM ne le rendant pas disponible librement, il faut le récupérer sur Sci-Hub <http://sci-hub.io/10.1145/1851182.1851192> (encore merci aux créateurs de ce service).

Merci à djanos <https://mastodon.tetaneutral.net/@djanos> pour ses nombreuses corrections sur la gestion de DCTCP dans Linux.