RFC 8289 : Controlled Delay Active Queue Management

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 6 janvier 2018

Date de publication du RFC : Janvier 2018

https://www.bortzmeyer.org/8289.html

Ah, la gestion des files d'attentes...Le cauchemar de plein d'étudiants en informatique. Et cela ne cesse pas quand ils deviennent ingénieurs et qu'il faut construire un routeur pour connecter des réseaux de capacités différentes, et qui aura donc besoin de files d'attente. Bref, dès qu'on n'a pas assez de ressources (et on n'en aura jamais assez), il faut optimiser ses files d'attente. Ce nouveau RFC décrit le mécanisme CoDel (mis en œuvre depuis un certain temps dans le noyau Linux) qui permet notamment de limiter le terrible, l'épouvantable "*bufferbloat*".

L'algorithme naïf pour gérer une file d'attente est le suivant (on prend le cas simple d'un routeur qui n'a que deux interfaces et une seule file d'attente dans chaque direction) : les paquets arrivent au routeur et sont mis dans la file gérée en FIFO. Dès que des ressources suffisantes sont disponibles pour envoyer un paquet (dès que l'interface de sortie est libre), on envoie le paquet et on le retire donc de la file. Si un paquet arrive quand la file est pleine, on le jette : TCP détectera cette perte, réduira son rythme d'envoi, et réémettra les données manquantes.

Avec cet algorithme simpliste, il y a une décision importante à prendre, la taille de la file. Le trafic sur l'Internet est tout sauf constant : des périodes de grand calme succèdent à des pics de trafic impressionnants. Si la file d'attente est trop petite, on ne pourra pas écluser ces pics, et on jettera des paquets. Ça tombe bien, le prix des mémoires baisse, on va pouvoir mettre des files plus grandes, non ? Eh bien non car arrive le fameux "*bufferbloat*". Si la file est trop grande, les paquets y séjourneront très longtemps, et on augmentera ainsi la latence <https://www.bortzmeyer.org/latence.html>, au grand dam des applications sensibles comme SSH ou les jeux en ligne. Bref, on est coincés, il n'y a pas de solution idéale. Pire, si on ne jette des paquets que lorsque la file est pleine, on risque de tomber dans le cas où l'équilibre se fait avec une file d'attente toujours pleine, et donc avec une mauvaise latence.

Bien sûr, il y a longtemps que les routeurs n'utilisent plus d'algorithme aussi naïf que celui présenté ici. Tout un travail a été fait sur l'AQM : voir par exemple les RFC 7567 [1] et RFC 8033. Mais le problème

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc7567.txt

de la file d'attente toujours pleine est toujours là. Sa première description est dans le RFC 896 en 1984. Plus récemment, on le trouve mentionné dans une présentation de Jim Gettys à l'IETF 80 <https://www.ietf.org/proceedings/80/slides/tsvarea-1.pdf> (« "*Bufferbloat : Dark Buffers in the Internet*" ») développé dans un article dans Communications of the ACM <https://cacm.acm.org/magazines/2012/1/144810-bufferbloat/fulltext> (Gettys a beaucoup fait pour la prise de conscience des dangers du "*bufferbloat*".).

Résoudre le problème de l'« obésité du tampon » ("*bufferbloat*") en réduisant la taille des files d'attentes ne serait pas une solution : les tampons sont là pour une bonne raison, pour absorber les problèmes brefs, lorsque, **temporairement**, on reçoit davantage de paquets que ce que l'on peut transmettre. Le fait que des files plus petites ne sont pas une solution a déjà été exposé dans le RFC 2309, dans « "*A Rant on Queues*" <http://www.pollere.net/Pdfdocs/QrantJul06.pdf> » de Van Jacobson, dans le rapport « "*RED in a Different Light*" <https://pdfs.semanticscholar.org/49d4/7813cc99fbcf5153628efpdf> » et dans l'article fondateur de CoDel, « "*Controlling Queue Delay*" <https://cacm.acm.org/magazines/2012/7/151223-controlling-queue-delay/abstract> » (article complet sur Sci-Hub, cherchez 10.1145/2209249.2209264). Le problème n'est pas tant la taille de la file en soi (ce n'est pas la taille qui compte), mais si c'est une « bonne » file ou une « mauvaise » file (au passage, si vous ne connaissez pas la différence entre le bon chasseur et le mauvais chasseur, ne ratez pas l'**indispensable** sketch des Inconnus <https://www.youtube.com/watch?v=QuGcoOJKXT8>).

CoDel est donc une vieille idée. Elle veut répondre, entre autres, aux critères suivants (section 1 du RFC) :

— Être « zéro-configuration » ("*parameterless*"), ce qui avait été un problème fréquent de solutions comme RED. (Mon opinion personnelle est que CoDel n'est pas réellement sans configuration, comme on le voit plus loin dans le RFC, mais il est certainement « configuration minimale ».) CoDel s'ajuste tout seul, une fois défini l'ordre de grandeur du RTT des paquets qui passeront par le routeur.
— Capable de différencier le mauvais chasseur du bon chasseur, euh, pardon, la mauvaise file d'attente de la bonne.
— Être simple à programmer, pour fonctionner aussi bien dans les processeurs limités des routeurs "*low cost*" que dans les ASIC (rapides, mais pas très souples) des routeurs haut de gamme.

Lorsque CoDel estime nécessaire de prendre des mesures (le trafic entre trop vite), il peut jeter les paquets, ou les marquer avec ECN (RFC 3168).

La section 2 de notre RFC définit la terminologie de CoDel. Parmi les termes importants :

— Temps de passage ("*sojourn time*") : le temps passé par le paquet dans la file d'attente. C'est la donnée de base de CoDel, qui va essayer de minimiser ce temps de passage.
— File persistante ("*standing queue*") : une file d'attente qui reste pleine trop longtemps, « trop » étant de l'ordre du RTT le plus élevé parmi les flots des paquets qui attendent dans la file.

Passons maintenant à une description de haut niveau de CoDel. Son but est de différencier la mauvaise file (qui ne fait qu'ajouter du retard d'acheminement des paquets) de la bonne. Une file d'attente se forme lorsqu'il y a un goulet d'étranglement, parce qu'un lien à forte capacité <https://www.bortzmeyer.org/capacite.html> se déverse dans un lien à faible capacité, ou bien parce que plusieurs liens convergent vers un lien ayant la capacité de seulement l'un d'eux. Une notion importante à ce sujet est celle de BDP, en gros le nombre d'octets en transit pour une connexion donnée, lorsque le débit atteint 100 % de la capacité. Imaginons une connexion TCP dont la fenêtre d'envoi est de 25 paquets (je sais bien que les fenêtres TCP se calculent en octets, pas en paquets, mais c'est le RFC qui fait cet abus de langage, pas moi) et où le BDP est de 20 paquets. En régime permanent, il y aura 5 paquets dans la file d'attente. Si la fenêtre est de 30 paquets, ce seront 10 paquets qui occuperont en permanence la file d'attente, augmentant encore le délai, alors que le débit ne changera pas (20 paquets arriveront par RTT). Au contraire, si on **réduit** la fenêtre à 20 paquets, la file d'attente sera vide, le délai sera réduit au minimum, alors que le débit n'aura pas changé. Ce résultat contre-intuitif montre que bourrer la connexion de paquets n'est pas forcément le meilleur moyen d'aller « vite ». Et que la taille

de la file ne renseigne pas sur le rythme d'envoi des données. Et enfin que les files pleines en permanence n'apportent que du délai. Dans le premier exemple, la file contenant cinq paquets tout le temps était clairement une **mauvaise file**. Un tampon d'entrée/sortie de 20 paquets est pourtant une bonne chose (pour absorber les variations brutales) mais, s'il ne se vide pas complètement ou presque en un RTT, c'est qu'il est mal utilisé. Répétons : **Les bonnes files se vident vite.**

CoDel comporte trois composants : un estimateur, un objectif et une boucle de rétroaction. La section 3 de notre RFC va les présenter successivement. Pour citer l'exposé de Van Jacobson à une réunion IETF <https://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf>, ces trois composants sont :

— *"a) Measure what you[Caractère Unicode non montré [2]]ve got"*
— *"b) Decide what you want"*
— *"c) If (a) isn[Caractère Unicode non montré]t (b), move it toward (b)"*

D'abord, l'estimateur. C'est la partie de CoDel qui observe la file d'attente et en déduit si elle est bonne ou mauvaise. Autrefois, la principale métrique était la taille de la file d'attente. Mais celle-ci peut varier très vite, le trafic Internet étant très irrégulier. CoDel préfère donc observer le temps de séjour dans la file d'attente. C'est une information d'autant plus essentielle qu'elle a un impact direct sur le vécu de l'utilisateur, via l'augmentation de la latence <https://www.bortzmeyer.org/latence.html>.

Bon, et une fois qu'on observe cette durée de séjour, comment en déduit-on que la file est bonne ou mauvaise ? Notre RFC recommande de considérer la durée de séjour **minimale**. Si elle est nulle (c'est-à-dire si au moins un paquet n'a pas attendu du tout dans la file, pendant la dernière période d'observation), alors il n'y a pas de file d'attente permanente, et la file est bonne.

Le RFC parle de « période d'observation ». Quelle doit être la longueur de cette période ? Au moins un RTT pour être sûr d'écluser les pics de trafic, mais pas moins pour être sûr de détecter rapidement les mauvaises files. Le RFC recommande donc de prendre comme longueur le RTT maximal de toutes les connexions qui empruntent ce tampon d'entrée/sortie.

Et, petite astuce d'implémentation (un routeur doit aller **vite**, et utilise souvent des circuits de calcul plus simples qu'un processeur généraliste), on peut calculer la durée de séjour minimale avec une seule variable : le temps écoulé depuis lequel la durée de séjour est inférieure ou supérieure au seuil choisi. (Dans le pseudo-code de la section 5, et dans le noyau Linux, c'est à peu près le rôle de `first_above_time`.)

Si vous aimez les explications avec images, il y en a plein dans cet excellent exposé à l'IETF <https://www.ietf.org/proceedings/84/slides/slides-84-tsvarea-4.pdf>.

Ensuite, l'objectif (appelé également référence) : il s'agit de fixer un objectif de durée de séjour dans la file. Apparemment, zéro serait l'idéal. Mais cela entrainerait des « sur-réactions », où on jetterait des paquets (et ralentirait TCP) dès qu'une file d'attente se forme. On va plutôt utiliser un concept dû à l'inventeur du datagramme, Leonard Kleinrock, dans « *"An Invariant Property of Computer Network Power"* <http://www.lk.cs.ucla.edu/data/files/Gail/power.pdf> », la « puissance » (*"power"*). En gros, c'est le débit divisé par le délai et l'endroit idéal, sur la courbe de puissance, est en haut (le maximum de débit, pour le minimum de délai). Après une analyse que je vous épargne, le RFC recommande de se fixer comme objectif entre 5 et 10 % du RTT.

Enfin, la boucle de rétroaction. Ce n'est pas tout d'observer, il faut agir. CoDel s'appuie sur la théorie du contrôle, pour un système MIMO (*"Multi-Input Multi-Output"*). Au passage, si vous ne comprenez

2. Car trop difficile à faire afficher par LaTeX

rien à la théorie du contrôle et notamment à la régulation PID, je vous recommande chaudement un article qui l'explique sans mathématiques <http://www.ferdinandpiette.com/blog/2011/08/implementer-un-pid-sans-faire-de-calculs/>. Placé à la fin de la file d'attente, au moment où on décide quoi faire des paquets, le contrôleur va les jeter (ou les marquer avec ECN) si l'objectif de durée de séjour est dépassé.

Passons maintenant à la section 4 du RFC, la plus concrète, puisqu'elle décrit précisement CoDel. L'algorithme a deux variables, `TARGET` et `INTERVAL` (ces noms sont utilisés tels quels dans le pseudo-code en section 5, et dans l'implémentation dans le noyau Linux). `TARGET` est l'objectif (le temps de séjour dans la file d'attente qu'on ne souhaite pas dépasser) et `INTERVAL` est la période d'observation. Ce dernier est également le seul paramètre de CoDel qu'il faut définir explicitement. Le RFC recommande 100 ms par défaut, ce qui couvre la plupart des RTT qu'on rencontre dans l'Internet, sauf si on parle à des gens très lointains ou si on passe par des satellites (cf. M. Dischinger, « "*Characterizing Residential Broadband Networks*" », dans les "*Proceedings of the Internet Measurement Conference*", San Diego, en 2007, si vous voulez des mesures). Cela permet, par exemple, de vendre des routeurs bas de gamme, genre point d'accès Wifi sans imposer aux acheteurs de configurer CoDel.

Si on est dans un environnement très différent de celui d'un accès Internet « normal », il peut être nécessaire d'ajuster les valeurs (CoDel n'est donc pas réellement « "*parameterless*" »). L'annexe A du RFC en donne un exemple, pour le cas du centre de données, où les latences sont bien plus faibles (et les capacités plus grandes) ; `INTERVAL` peut alors être défini en microsecondes plutôt qu'en millisecondes.

`TARGET`, lui, va être déterminé dynamiquement par CoDel. Une valeur typique sera aux alentours de 5 ms (elle dépend évidemment de `INTERVAL`) : si, pendant une durée égale à `INTERVAL`, les paquets restent plus longtemps que cela dans la file d'attente, c'est que c'est une mauvaise file, on jette des paquets. Au passage, dans le noyau Linux, c'est dans `codel_params_init` que ces valeurs sont fixées :

```
  params->interval = MS2TIME(100);
params->target = MS2TIME(5);
```

Les programmeurs apprécieront la section 5, qui contient du pseudo-code, style C++, mettant en œuvre CoDel. Le choix de C++ est pour profiter de l'héritage, car CoDel est juste une dérivation d'un classique algorithme "*tail-drop*". On peut donc le programmer sous forme d'une classe qui hérite d'une classe `queue_t`, plus générale.

De toute façon, si vous n'aimez pas C++, vous pouvez lire le code source du noyau Linux, qui met en œuvre CoDel depuis longtemps (fichier `net/sched/sch_codel.c`, également accessible via le Web <https://github.com/torvalds/linux/blob/master/net/sched/sch_codel.c>).

(Pour comprendre les exemples de code, notez que `packet_t*` pointe vers un descripteur de paquet, incluant entre autres un champ, `tstamp`, qui stocke un temps, et que le type `time_t` est exprimé en unités qui dépendent de la résolution du système, sachant que la milliseconde est suffisante, pour du trafic Internet « habituel ».) Presque tout le travail de CoDel est fait au moment où le paquet sort de la file d'attente. À l'entrée, on se contente d'ajouter l'heure d'arrivée, puis on appelle le traitement habituel des files d'attente :

```
void codel_queue_t::enqueue(packet_t* pkt)
   {
       pkt->tstamp = clock();
       queue_t::enqueue(pkt);
   }
```

Le gros du code est dans le sous-programme `dodequeue` qui va déterminer si le paquet individuel vient de nous faire changer d'état :

```
dodequeue_result codel_queue_t::dodequeue(time_t now)
   {
   ...
       // Calcul de *la* variable importante, le temps passé dans la
       // file d'attente
       time_t sojourn_time = now - r.p->tstamp;
       // S'il est inférieur à l'objectif, c'est bon
       if (sojourn_time_ < TARGET || bytes() <= maxpacket_) {
           first_above_time_ = 0;
       } else {
           // Aïe, le paquet est resté trop longtemps (par rapport à TARGET)
           if (first_above_time_ == 0) {
       // Pas de panique, c'est peut-être récent, attendons
       // INTERVAL avant de décider
                first_above_time_ = now + INTERVAL;
           } else if (now >= first_above_time_) {
               // La file ne se vide pas : jetons le paquet
               r.ok_to_drop = true;
           }
       }
       return r;
   }
```

Le résultat de `dodequeue` est ensuite utilisé par `dequeue` qui fait se fait quelques réflexions supplémentaires avant de jeter réellement le paquet.

Ce code est suffisamment simple pour pouvoir être mis en œuvre dans le matériel, par exemple ASIC ou NPU.

L'annexe A du RFC donne un exemple de déclinaison de CoDel pour le cas d'un centre de données, où les RTT typiques sont bien plus bas que sur l'Internet public. Cette annexe étudie également le cas où on applique CoDel aux files d'attente des serveurs, pas uniquement des routeurs, avec des résultats spectaculaires :

— Dans le serveur, au lieu de jeter le paquet bêtement, on prévient directement TCP qu'il doit diminuer la fenêtre,
— Le RTT est connu du serveur, et peut donc être mesuré au lieu d'être estimé (on en a besoin pour calculer `INTERVAL`),
— Les paquets de pur contrôle (`ACK`, sans données) ne sont jamais jetés, car ils sont importants (et de petite taille, de toute façon).

Sur l'Internet public, il serait dangereux de ne jamais jeter les paquets de pur contrôle, ils pourraient être envoyés par un attaquant. Mais, dans le serveur, aucun risque.

Également à lire : un article de synthèse sur le blog de l'IETF `<https://www.ietf.org/blog/codel-improved-networking-through-adaptively-managed-router-queues/>`.