RFC 8033 : Proportional Integral Controller Enhanced (PIE): A Lightweight Control Scheme to Address the Bufferbloat Problem

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 1 mars 2017

Date de publication du RFC : Février 2017

https://www.bortzmeyer.org/8033.html

Mais c'est quoi, ce "*bufferbloat*" (obésité du tampon ?) et pourquoi est-ce mauvais ? Le "*bufferbloat*" désigne la tendance à mettre dans les routeurs des tampons d'entrée/sortie de plus en plus gros. Cette tendance vient de la baisse du prix des mémoires, et d'un désir de pouvoir encaisser les brusques pics de trafic ("*bursts*") qui sont fréquents sur l'Internet. Mais le "*bufferbloat*" a une conséquence négative : la latence <https://www.bortzmeyer.org/latence.html> augmente, puisque le paquet risque de devoir attendre longtemps dans un tampon qui, une fois rempli, aura du mal à se vider. Ce RFC propose donc un mécanisme de gestion des files d'attente, PIE ("*Proportional Integral controller Enhanced*") où le routeur surveille la latence des paquets dans les tampons d'entrée/sortie, et jette des paquets, même si le tampon n'est pas plein, pour limiter la latence. Le paquet perdu dira en effet aux émetteurs de ralentir.

La latence <https://www.bortzmeyer.org/latence.html> est particulièrement à surveiller dans le cas d'applications fortement interactives comme les jeux en ligne ou la vidéoconférence. On cherche donc à diminuer la latence, pour fournir une meilleure qualité de service aux utilisateurs. PIE a fait l'objet d'analyses théoriques, de simulations, puis de mise en œuvre dans le noyau Linux, et semble aujourd'hui une solution intéressante. PIE est une solution purement locale au routeur, et ne pose donc pas de problèmes d'interopérablité : les autres routeurs avec lesquels on parle n'ont pas besoin de participer.

L'un des problèmes centraux de l'Internet a toujours été la congestion. Les paquets arrivent quand ils veulent, et peuvent dépasser la capacité <https://www.bortzmeyer.org/capacite.html> du réseau. Deux solutions pour un routeur, jeter les paquets (IP est prévu pour cela, il travaille en mode datagramme), et attendre que les couches supérieures comme TCP s'en aperçoivent et ralentissent, ou bien deuxième solution, mettre les paquets dans un tampon, en attendant de pouvoir les envoyer. Ce tampon va permettre de lisser un trafic Internet qui est très irrégulier. En pratique, les deux solutions

doivent être déployées : le tampon a une taille finie et, s'il est plein, il faut bien se résigner à jeter des paquets.

Comme la perte de paquets entraine un ralentissement du transfert de données (TCP va automatiquement diminuer le débit), il existe une forte demande pour limiter cette perte. La baisse des prix des mémoires permet de satisfaire cette demande, en augmentant la taille des tampons. (Voir le site Web consacré au « bufferbloat » <https://www.bufferbloat.net/>, qui contient notamment une bonne introduction au problème <https://www.bufferbloat.net/projects/bloat/wiki/Introduction/>.)

L'effet pervers de cette augmentation de taille est que les protocoles comme TCP, ne voyant pas de perte de paquets, vont continuer à augmenter leur débit, et envoyer plein de paquets jusqu'à ce que, le tampon étant plein, le routeur commence à jeter des paquets, calmant TCP. Mais, à ce moment, il est trop tard, le tampon est plein et risque de rester plein longtemps, l'émetteur continuant à envoyer des paquets, même si c'est à un rythme réduit. Les paquets vont donc patienter dans le tampon, augmentant la latence <https://www.bortzmeyer.org/latence.html>. Et plus le tampon est grand, plus on aggrave la latence. On est donc passé de Charybde en Scylla : pour éviter les pertes de paquets, qui diminuent le débit, on a augmenté la latence. (On voit d'ailleurs que la notion de performance, dans les réseaux, est une notion compliquée. C'est pour cela que des termes flous et passe-partout comme « vitesse » ne devraient pas être employés.)

Un système de gestion de la file d'attente (AQM) va permettre de mieux contrôler le problème, en essayant de faire en sorte que les pics soudains d'activité puissent passer, tout en limitant la latence pour les transferts de longue durée. Un exemple de mécanisme d'AQM est RED, initialement proposé dans le RFC 2309 [1] il y a dix-huit ans. RED a deux limites, il nécessite un réglage manuel de ses paramètres, et il agit sur la longueur de la file d'attente, pas sur la latence. C'est entre autre pour cela que le RFC 7567 avait demandé à ce que de nouveaux mécanismes d'AQM soient développés.

L'algorithme de ce RFC, PIE, se veut, comme RED, simple et facile à mettre en œuvre. Comme RED, son principal moyen d'action est de jeter, de manière partiellement aléatoire, des paquets avant qu'ils ne soient mis dans la file d'attente. Contrairement à RED, il agit sur la latence, pas sur la longueur de la file d'attente.

Les objectifs de PIE sont décrits dans la section 3 du RFC :

— Contrôler la latence, le paramètre qui est réellement important pour les applications,
— Essayer d'utiliser le réseau au mieux de sa capacité (si on jette trop de paquets, TCP va tellement ralentir que, certes, les tampons seront vides et la latence excellente, mais le réseau ne sera plus utilisé à fond),
— Simple à programmer et déployer (pas de réglage manuel des paramètres).

La section 4 du RFC décrit PIE, et c'est la section à lire si vous voulez mettre en œuvre PIE dans un routeur, ou simplement le comprendre complètement. L'algorithme effectue trois tâches :

— Jeter des paquets aléatoirement, avec une certaine probabilité, lors de l'arrivée dans la file d'attente,
— Mettre à jour automatiquement en permanence cette probabilité,
— Calculer la latence (puisque c'est elle qu'on veut minimiser).

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc2309.txt

La description complète originale figure dans l'article de Pan, R., Natarajan, P. Piglione, C., Prabhu, M.S., Subramanian, V., Baker, F. Steeg et B. V., « "*PIE : A Lightweight Control Scheme to Address the Bufferbloat Problem*" <https://www.researchgate.net/publication/261134127_PIE_A_lightweight_control_scheme_to_address_the_bufferbloat_problem?origin=mail> » en 2013. Cet algorithme suit les principes de stabilité de théorie du contrôle.

Dans cette section 4, notre RFC présente l'algorithme PIE sous forme de texte et de pseudo-code. La première tâche (section 4.1), jeter les paquets entrants selon une certaine probabilité (`PIE->drop_prob_`) va s'exprimer :

```
//Safeguard PIE to be work conserving
if ( (PIE->qdelay_old_ < QDELAY_REF/2 && PIE->drop_prob_ < 0.2)
|| (queue_.byte_length() <= 2 * MEAN_PKTSIZE) ) {
     return ENQUE;
else
    randomly drop the packet with a probability PIE->drop_prob_.
```

La première branche du `if` est là pour éviter du travail inutile : si la probabilité de jeter un paquet est faible, ou bien si la file d'attente est loin d'être pleine (moins de deux paquets en attente), ou bien si la latence est bien plus faible que la latence visée, dans ces cas, on le jette rien. C'est le fonctionnement idéal du routeur, lorsque la congestion n'est qu'une menace lointaine.

La deuxième tâche, calculer automatiquement la probabilité de jeter un paquet, est plus délicate (section 4.2). Il faut connaitre la latence mais aussi la tendance (est-ce que la latence tend à diminuer ou bien à augmenter). C'est ce qu'on nomme le contrôleur "*Proportional Integral*" qui a donné son nom à l'algorithme PIE. La formule de base (voir le pseudo-code complet dans le RFC, notamment dans l'annexe A) est que la probabilité est la latence (`current_qdelay`) multipliée par un coefficient (`alpha`), augmentée de la différence entre la latence actuelle et la latence précédente (et, donc, si la latence diminue, la probabilité sera diminuée) :

```
p = alpha*(current_qdelay-QDELAY_REF) +
            beta*(current_qdelay-PIE->qdelay_old_);
```

Et la troisième tâche, le calcul de la latence, est fait en suivant la loi de Little (section 4.3) :

```
current_qdelay = queue_.byte_length()/dequeue_rate;
```

Cette formule est une estimation de la latence. On peut aussi la mesurer directement (mais cela fait plus de travail pour le routeur), par exemple en ajoutant une estampille temporelle aux paquets entrants et en la lisant à la sortie.

Ce pseudo-code n'est encore qu'une approximation du vrai algorithme. L'un des gros problèmes de tout système de gestion de la file d'attente est que le trafic Internet est sujet à de brusques pics où un grand nombre de paquets arrive en peu de temps. Cela va remplir la file et augmenter la latence, mais cela ne veut pas dire qu'il faille subitement augmenter la probabilité d'abandon de paquets (section 4.4). Donc, la première tâche, jeter certains paquets, devient :

```
if PIE->burst_allowance_ > 0 enqueue packet;
else randomly drop a packet with a probability PIE->drop_prob_.

if (PIE->drop_prob_ == 0 and current_qdelay < QDELAY_REF/2 and PIE->qdelay_old_ < QDELAY_REF/2)
    PIE->burst_allowance_ = MAX_BURST;
```

Et dans la seconde, le calcul de la probabilité d'abandon de paquets, on ajoute :

```
PIE->burst_allowance_ = max(0,PIE->burst_allowance_ - T_UPDATE);
```

Cette fois, on a un PIE complet. Mais on peut, optionnellement, y ajouter certains éléments (section 5 du RFC). Le plus évident est, au lieu de jeter le paquet, ce qui fait qu'il aura été émis et transmis par les routeurs amont pour rien, de marquer les paquets avec ECN (RFC 3168). La première tâche regarde donc si le flot de données gère ECN et utilise cette possibilité dans ce cas, au lieu de jeter aveuglément :

```
if PIE->drop_prob_ < mark_ecnth && ecn_capable_packet:
    mark packet;
else:
    drop packet;
```

Le trafic réseau varie beaucoup dans le temps. La plupart du temps, si le réseau est bien dimensionné, il n'y a pas de problème et il serait dommage que PIE jette au hasard des paquets quand on n'est dans cette phase heureuse. Un autre ajout utile à PIE est donc une désctivation automatique quand la file d'attente est peu remplie. Un des avantages de couper complètement PIE (par rapport à simplement décider de ne pas jeter les paquets) est de gagner du temps dans le traitement des paquets.

Pour réactiver PIE quand la congestion commence, c'est un peu plus compliqué. Si PIE est coupé, il n'y a plus de calcul de la latence, et on ne peut donc pas utiliser une augmentation de la latence pour décider de remettre PIE en marche. Le RFC suggère de remettre PIE en route dès qu'on passe au-dessus d'un tiers d'occupation de la file d'attente.

Autre question délicate, les problèmes que crée le hasard. Par défaut, PIE prend ses décisions en jetant les dés. Si la latence est importante, indiquant qu'on approche de la congestion, PIE jette des paquets au hasard. Mais le hasard n'est pas prévisible (évidemment). Et il ne mène pas à une répartition uniforme des pertes de paquets. Il se peut qu'aucun paquet ne soit jeté pendant longtemps, ce qui fait que le routeur ne réagira pas à l'augmentation de la latence. Mais il se peut aussi qu'un massacre de paquets se produise à certains moments. L'utilisation du hasard mène forcément à des « séries noires » (ou à des « séries blanches »). Notre RFC propose donc un mécanisme (optionnel) de « dé-hasardisation », où un nouveau paramètre augmente avec la probabilité d'abandon de paquet, et est remis à zéro lorsqu'on jette un paquet. La décision de laisser tomber un paquet n'est prise que lorsque ce paramètre est entre deux valeurs pré-définies.

La section 6 du RFC se penche sur les problèmes concrets de mise en œuvre (programmeurs, on pense à vous). PIE peut être mis en œuvre en logiciel ou bien en matériel (sur beaucoup de routeurs, la

mise en file d'attente est typiquement « plus logicielle » que le retrait de la file). PIE est simple, et peut être programmé de manière très économique (ou plus coûteuse si on met une estampille temporelle à chaque paquet, ce qui permet de mieux mesurer la latence, mais nécessite davantage de mémoire).

La deuxième tâche de PIE, recalculer la probabilité d'abandon, se fait typiquement en parallèle avec le traitement de la file d'attente. Vu le rythme d'entrée et de sortie des paquets dans un routeur moderne, ce sont des milliers de paquets qui sont passés entre deux recalculs. Le routeur ne pourra donc pas réagir instantanément.

Comme tous les bons algorithmes, PIE est évidemment plombé par un brevet, en l'occurrence deux brevets de Cisco. Cette entreprise a promis une licence gratuite et sans obligations (mais avec la classique clause de représailles, annulant cette licence si quelqu'un essaie d'utiliser ses brevets contre Cisco).

Aujourd'hui, Linux, FreeBSD (voir la page Web du projet <http://caia.swin.edu.au/freebsd/aqm/>) et d'autres mettent en œuvre PIE.