RFC 6555 : Happy Eyeballs: Success with Dual-Stack Hosts

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 avril 2012

Date de publication du RFC : Avril 2012

https://www.bortzmeyer.org/6555.html

Une question récurrente qu'IPv6 pose aux développeurs d'application et aux administrateurs système qui déploieront ces applications ensuite est celle d'une machine accessible à la fois en IPv4 et IPv6. Si le pair cherchant à joindre cette machine croit avoir IPv6 mais que sa connectivité ne marche pas, ou mal, l'application arrivera-t-elle à se rabattre en IPv4 très vite ou bien imposera-t-elle à l'utilisateur un long délai avant de détecter enfin le problème ? Cette question est connue comme « le bonheur des globes oculaires <https://www.bortzmeyer.org/globes-oculaires-heureux.html> » (les dits globes étant les yeux de l'utilisateur qui attend avec impatience la page d'accueil de YouPorn) et ce RFC spécifie les exigences pour l'algorithme de connexion du client, afin de rendre les globes oculaires le plus heureux possible. (Il a depuis été remplacé par un RFC plus général, ne se limitant pas à IPv6, le RFC 8305[1].)

La section 1 rappelle les données du problème : on veut évidemment que cela marche aussi bien en IPv6 qu'en IPv4 (pas question d'accepter des performances inférieures) or, dans l'état actuel du déploiement d'IPv6, bien des sites ont une connexion IPv6 totalement ou partiellement cassée. Si un serveur a IPv4 et IPv6 et que son client n'a qu'IPv4, pas de problème. Mais si le client a IPv6, tente de l'utiliser, mais que sa connexion est plus ou moins en panne, ses globes oculaires vont souffrir d'impatience. Certains fournisseurs (notamment Google) ont résolu le problème en n'annonçant leur connectivité IPv6 (via le DNS et les enregistrements AAAA) qu'à certains réseaux, configurés manuellement après examen de la qualité effective de leur connectivité. Cette approche, nommée « "*whitelisting*" » ne passe évidemment pas à l'échelle. Même pour Google, c'est un travail colossal que d'évaluer le réseau de tous les sites Internet.

(À noter que deux lecteurs m'ont fait remarquer que YouPorn n'est pas un bon exemple, puisqu'il ne publie pas d'adresse IPv6. C'est vrai et cela montre que mes lecteurs connaissent bien YouPorn.)

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc8305.txt

La bonne solution est donc que l'application elle-même gère le problème (ou, sinon l'application elle-même, la bibliothèque logicielle qu'elle utilise et où se trouve la fonction de connexion). Il existe plusieurs algorithmes pour cela <https://www.bortzmeyer.org/globes-oculaires-heureux.html>, déjà largement déployés. Ce RFC normalise les caractéristiques que doivent avoir ces algorithmes. Si on suit ce RFC, le trafic (IP et DNS) va légèrement augmenter (surtout si la connectivité IPv6 marche mal ou pas du tout) mais la qualité du vécu de l'utilisateur va être maintenue, même en présence de problèmes, ce qui compense largement. Autrement, il existerait un risque élevé que certains utilisateurs coupent complètement IPv6, plutôt que de supporter ces problèmes de délai de connexion.

La section 3 revient dans l'histoire, pour montrer que le problème n'est pas nouveau. En 1994, le RFC 1671 disait « "*The dual-stack code may get two addresses back from DNS ; which does it use ? During the many years of transition the Internet will contain black holes.*" » (Au passage, cela montre la stupidité de la légende, fréquemment entendue, comme quoi les problèmes de la transition IPv4-¿IPv6 n'auraient pas été étudiés à l'avance.) Tout le reste de notre RFC 6555 est consacré à répondre à cette question. La cible principale est composée des protocoles de transport avec connexion (TCP, SCTP), les protocoles sans connexion comme UDP soulevant d'autres questions (s'ils ont une sémantique requête/réponse, comme dans ICE, les algorithmes de ce RFC peuvent être utilisés).

Le RFC rejette l'idée d'utiliser des noms différents pour les deux familles (comme `www.ipv6.example.com`), car cela complique le partage de noms (envoi d'un URL à un copain, par exemple).

Donc, on a un nom de machine qu'on veut contacter, mettons `www.example.com`, avec deux adresses associées, une par famille (voir les sections 5.3 et 5.4 pour le cas avec plus de deux adresses). Avec l'algorithme naïf qu'utilisent certains logiciels, et une connexion IPv6 qui marche mal, voici la séquence d'évenements :

— L'initiateur de la connexion utilise le DNS pour demander les enregistrements A (adresse IPv4) et AAAA (IPv6). (Notez qu'il n'existe pas de type de requête DNS pour avoir les deux enregistrements d'un coup, il faut donc deux requêtes. Voir aussi la section 5.4.)
— Il récupère `192.0.2.1` et `2001:db8::1`.
— Il tente IPv6 (sur Linux, l'ordre des essais est réglable dans `/etc/gai.conf`). L'initiateur envoie un paquet TCP `SYN` à `2001:db8::1`.
— Pas de réponse (connexion IPv6 incorrecte). L'initiateur réessaie, deux fois, trois fois, faisant ainsi perdre de nombreuses secondes.
— L'initiateur renonce, il passe à IPv4 et envoie un paquet TCP `SYN` à `192.0.2.1`.
— Le répondeur envoie un `SYN+ACK` en échange, l'initiateur réplique par un `ACK` et la connexion TCP est établie.

Le problème de cet algorithme naïf est donc la longue attente lors des essais IPv6. On veut au contraire un algorithme qui bascule rapidement en IPv4 lorsqu'IPv6 ne marche pas, sans pour autant gaspiller les ressources réseau en essayant par exemple toutes les adresses en même temps.

L'algorithme recommandé (section 4, cœur de ce RFC) aura donc la tête :

— L'initiateur de la connexion utilise le DNS pour demander les enregistrements A (adresse IPv4) et AAAA (IPv6).
— Il récupère `192.0.2.1` et `2001:db8::1`. Il sait donc qu'il a plusieurs adresses, de famille différente.
— Il tente IPv6. L'initiateur envoie un paquet TCP `SYN` à `2001:db8::1`, avec un très court délai de garde.
— Pas de réponse ? L'initiateur passe à IPv4 tout de suite. Il envoie un paquet TCP `SYN` à `192.0.2.1`.
— Le répondeur envoie un `SYN+ACK` en échange, l'initiateur réplique par un `ACK` et la connexion TCP est établie.

Si le répondeur réagit à une vitesse normale en IPv6, la connexion sera établie en IPv6. Sinon, on passera vite en IPv4, et l'utilisateur humain ne s'apercevra de rien. Naturellement, si le DNS n'avait rapporté qu'une seule adresse (v4 ou v6), on reste à l'algorithme traditionnel (« essayer, patienter, ré-essayer »).

La section 4 précise ensuite les exigences auxquelles doit obéir l'algorithme. D'abord, il est important de ne pas tester IPv4 tout de suite. Les premiers algorithmes « bonheur des globes oculaires » envoyaient les deux paquets `SYN` en même temps, gaspillant des ressources réseau et serveur. Ce double essai faisait que les équipements IPv4 du réseau avaient autant de travail qu'avant, alors qu'on aurait souhaité les retirer du service petit à petit. En outre, ce test simultané fait que, dans la moitié des cas, la connexion sera établie en IPv4, empêchant de tirer profit des avantages d'IPv6 (cf. RFC 6269). Donc, on **doit** tester en IPv6 d'abord, sauf si on se souvient des tentatives précédentes (voir plus loin la variante « avec état ») ou bien si l'administrateur système a délibérement configuré la machine pour préférer IPv4.

L'avantage de cet algorithme « IPv6 d'abord puis rapidement basculer en IPv4 » est qu'il est sans état : l'initiateur n'a pas à garder en mémoire les caractéristiques de tous ses correspondants. Mais son inconvénient est qu'on recommence le test à chaque connexion. Il existe donc un algorithme avec état, où l'initiateur peut garder en mémoire le fait qu'une machine (ou bien un préfixe entier) a une adresse IPv6 mais ne répond pas aux demandes de connexion de cette famille. Le RFC recommande toutefois de réessayer IPv6 au moins toutes les dix minutes, pour voir si la situation a changé. De plus, un changement de connectivité (détecté par le DNA des RFC 4436 ou RFC 6059) doit entraîner un vidage complet de l'état (on doit oublier ce qu'on a appris, qui n'est plus pertinent).

Une conséquence de l'algorithme recommandé est que, dans certains cas, les **deux** connexions TCP (v4 et v6) seront établies (si le `SYN` IPv6 voyage lentement et que la réponse arrive après que l'initiateur de la connexion se soit impatienté et soit passé à IPv4). Cela peut être intéressant dans certains cas rares, mais le RFC recommande plutôt d'abandonner la connexion perdante (la deuxième). Autrement, cela pourrait entraîner des problèmes avec, par exemple, les sites Web qui lient un "*cookie*" à l'adresse IP du client, et seraient surpris de voir deux connexions avec des adresses différentes.

Voilà, l'essentiel du RFC est là. La section 5 donne quelques détails en plus. Par exemple, l'algorithme des globes oculaires heureux est astucieux, mais tend à masquer les problèmes. Si un site Web publie les deux adresses mais que sa connectivité IPv6 est défaillante, aucun utilisateur ne lui signalera puisque, pour eux, tout va bien. Il est donc recommandé que le logiciel permette de débrayer cet algorithme, afin de tester la connectivité avec seulement v4 ou seulement v6, ou bien que le logiciel indique quelque part ce qu'il a choisi, pour mieux identifier d'éventuels problèmes v6.

D'autre part, vous avez peut-être remarqué que j'ai utilisé des termes vagues (« un certain temps ») pour parler du délai entre, par exemple, le premier `SYN` IPv6 et le premier `SYN` IPv4. La section 5.5 donne des idées quantitatives en suggérant entre 150 et 250 ms entre deux essais (les navigateurs actuels sont plutôt du côté de 300 ms). C'est quasiment imperceptible à un utilisateur humain devant son navigateur Web, tout en évitant de surcharger le réseau inutilement. Les algorithmes avec état ont le droit d'être plus impatients, puisqu'ils peuvent se souvenir des durées d'établissement de connexion précédents.

Autre problème pratique, l'interaction avec la politique de sécurité des navigateurs Web (RFC 6454). Pour limiter les risques d'attaque par changement DNS <`https://www.bortzmeyer.org/dns-rebinding-pinning.html`> (croire qu'on se connecte au même serveur HTTP alors que c'est en fait un autre), les navigateurs ne doivent pas changer de famille d'adresse (v4 en v6 ou réciproquement) une fois la session commencée.

Enfin, les implémentations. La section 6 donne les exemples de Firefox et Chrome, très proches de ce que décrit le RFC. D'autres possibilités sont disponibles en ligne, en Erlang <`http://www.viagenie.`

ca/news/index.html#happy_eyeballs_erlang> ou en C <http://www.isc.org/community/blog/201101/how-to-connect-to-a-multi-homed-server-over-tcp>. Une version pour Go avait été affichée en <http://www.pastie.org/1528545> (en voici une sauvegarde locale (en ligne sur https://www.bortzmeyer.org/files/multi-connect.go)).

Enfin, pour un exemple de test du malheur des globes oculaires, voir « *"Experiences of host behavior in broken IPv6 networks"* <http://www.ietf.org/proceedings/80/slides/v6ops-12.pdf> ». Ou bien « *"Evaluating the Effectiveness of Happy Eyeballs"* <https://labs.ripe.net/Members/vaibhav_bajpai/evaluating-the-effectiveness-of-happy-eyeballs> » qui contient d'intéressantes mesures.