RFC 8305 : Happy Eyeballs Version 2: Better Connectivity Using Concurrency

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 21 décembre 2017

Date de publication du RFC : Décembre 2017

https://www.bortzmeyer.org/8305.html

Une machine connectée à l'Internet et répondant aux requêtes venues du réseau a souvent plusieurs adresses IP pour son nom. C'est parfois une adresse IPv4 et une IPv6 mais cela peut aussi être plusieurs adresses IPv6, ou bien un mélange en proportions quelconques. Les développeurs d'application et les administrateurs système qui déploieront ces applications ensuite, ont un choix difficile si certaines de ces adresses marchent et d'autres pas (ou mal). Si les différentes adresses IP de cette machine passent par des chemins différents, certains marchant et d'autres pas, l'application arrivera-t-elle à se rabattre sur une autre adresse 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 PornHub) et ce RFC spécifie les exigences pour l'algorithme de connexion du client. En les suivant, les globes oculaires seront heureux. Il s'agit de la version 2 de l'algorithme, bien plus élaborée que la version 1 qui figurait dans le RFC 6555 [1].

La section 1 rappelle les données du problème : on veut évidemment que cela marche aussi bien en IPv6 (RFC 8200) 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, ou simplement sous-optimale, ses globes oculaires vont souffrir d'impatience. On peut aussi noter que le problème n'est pas spécifique à IPv6 : du moment que la machine visée a plusieurs adresses, qu'elles soient IPv4 ou IPv6, le risque que certaines des adresses ne marchent pas (ou moins bien) existe, et l'algorithme des globes oculaires heureux doit être utilisé. (C'est un des gros changements avec le précédent RFC, le RFC 6555, qui n'envisageait que le cas IPv6.)

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6555.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 depuis des années. On peut donc se baser sur l'expérience pour spécifier ces algorithmes. 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 cible principale de notre RFC 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).

Donc, on a un nom de machine qu'on veut contacter, mettons `www.example.com`, avec plusieurs adresses associées, peut-être de familles (v4 et v6) différentes. Prenons une machine ayant une seule adresse IPv4 et une seule adresse IPv6, avec une connexion IPv6 qui marche mal. Avec l'algorithme naïf qu'utilisent encore certains logiciels voici la séquence d'évenements traditionnelle :

— 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 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 (sections 3 à 5, cœur de ce RFC) aura donc l'allure suivante :

— 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'algorithme du RFC est de toute façon facilement adaptable à des cas où IPv4 est prioritaire). L'initiateur envoie un paquet TCP `SYN` à `2001:db8::1`, avec un très court délai de garde.
— Pas de réponse quasi-immédiate ? L'initiateur passe à IPv4 rapidement. 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 »).

Maintenant, les détails. D'abord, le DNS (section 3 de notre RFC). Pour récupérer les adresses appartenant aux deux familles (IPv4 et IPv6), il faut envoyer deux requêtes, de type A et AAAA. Pas de délai entre les deux, et le AAAA en premier, recommande le RFC. Notez qu'il n'existe pas de type de requête DNS pour avoir les deux enregistrements d'un coup, il faut donc deux requêtes.

Il ne faut pas attendre d'avoir la réponse aux deux avant de commencer à tenter d'établir une connexion. En effet, certains pare-feux configurés avec les pieds bloquent les requêtes AAAA, qui vont finir par "*timeouter*". Du point de vue du programmeur, cela signifie qu'il faut faire les deux requêtes DNS dans des fils différents (ou des "*goroutines*" différentes en Go), ou bien, utiliser une API asynchrone, comme getdns `<https://getdnsapi.net/>`. Ensuite, si on reçoit la réponse AAAA mais pas encore de A, on essaye tout de suite de se connecter, si on a la réponse A, on attend quelques millisecondes la réponse AAAA puis, si elle ne vient pas, tant pis, on essaie en IPv4. (La durée exacte de cette attente est un des paramètres réglables de l'algorithme. Il se nomme "*Resolution Delay*" et sa valeur par défaut recommandée est de 50 ms.)

À propos de DNS, notez que le RFC recommande également de privilégier IPv6 pour le transport des requêtes DNS vers les résolveurs (on parle bien du transport des paquets DNS, pas du type des données demandées). Ceci dit, ce n'est pas forcément sous le contrôle de l'application.

Une fois récupérées les adresses, on va devoir les trier selon l'ordre de préférence. La section 4 décrit comment cela se passe. Rappelons qu'il peut y avoir plusieurs adresses de chaque famille, pas uniquement une v4 et une v6, et qu'il est donc important de gérer une liste de toutes les adresses reçues (imaginons qu'on ne récupère que deux adresses v4 et aucune v6 : l'algorithme des globes oculaires heureux est quand même crucial car il est parfaitement possible qu'une des adresses v4 ne marche pas).

Pour trier, le RFC recommande de suivre les règles du RFC 6724, section 6. Si le client a un état (une mémoire des connexions précédentes, ce qui est souvent le cas chez les clients qui restent longtemps à tourner, un navigateur Web, par exemple), il peut ajouter dans les critères de tri le souvenir des succès (ou échecs) précédents, ainsi que celui des RTT passés. Bien sûr, 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).

Dernier détail sur le tri : il faut mêler les adresses des deux familles. Imaginons un client qui récupère trois adresses v6 et trois v4, client qui donne la priorité à IPv4, mais dont la connexion IPv4 est défaillante. Si sa liste d'adresses à tester comprend les trois adresses v4 en premier, il devra attendre trois essais avant que cela ne marche. Il faut donc plutôt créer une liste {une adressse v4, une adresse v6, une adresse v4…}. Le nombre d'adresses d'une famille à inclure avant de commencer l'autre famille est le paramètre "*First Address Family Count*", et il vaut un par défaut.

Enfin, on essaie de se connecter en envoyant des paquets TCP SYN (section 5). 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.

Après chaque essai, on attend pendant une durée paramétrable, "*Connection Attempt Delay*", 250 ms par défaut (bornée par les paramètres "*Minimum Connection Attempt Delay*", 100 ms par défaut, qu'on ne devrait jamais descendre en dessous de 10 ms, et "*Maximum Connection Attempt Delay*", 2 s par défaut).

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 (cf. plus haut), 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 re-essayer IPv6 au moins toutes les dix minutes, pour voir si la situation a changé.

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.

La section 9 du RFC rassemble quelques derniers problèmes pratiques. Par exemple, notre algorithme des globes oculaires heureux ne prend en compte que l'établissement de la connexion. Si une adresse ne marche pas du tout, il choisira rapidement la bonne. Mais si une adresse a des problèmes de MTU et pas l'autre, l'établissement de la connexion, qui ne fait appel qu'aux petits paquets TCP `SYN`, se passera bien alors que le reste de l'échange sera bloqué. Une solution possible est d'utiliser l'algorithme du RFC 4821.

D'autre part, l'algorithme ne tient compte que de la possibilité d'établir une connexion TCP, ce qui se fait typiquement uniquement dans le noyau du système d'exploitation du serveur. L'algorithme ne garantit pas qu'une application écoute, et fonctionne.

Parmi les problèmes résiduels, notez que l'algorithme des globes oculaires heureux est astucieux, mais tend à masquer les problèmes (section 9.3). 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 l'opérateur fasse des tests de son côté pour repérer les problèmes (le RFC 6555 recommandait 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.)

Pour le délai entre le premier `SYN` IPv6 et le premier `SYN` IPv4, la section 5 donne des idées quantitatives en suggérant 250 ms entre deux essais. C'est conçu pour être 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.

Notez que les différents paramètres réglables indiqués ont des valeurs par défaut, décrites en section 8, et qui ont été déterminées empiriquement.

Si vous voulez une meilleure explication de la version 2 des globes oculaires heureux, il y a cet exposé au RIPE <https://ripe75.ripe.net/presentations/129-ipv6-hev2-v2.pdf>.

Enfin, les implémentations. Notez que les vieilles mises en œuvre du RFC 6555 (et présentées à la fin de mon précédent article) sont toujours conformes à ce nouvel algorithme, elles n'en utilisent simplement pas les raffinements. Les versions récentes de macOS (Sierra) et iOS (10) mettent en œuvre notre RFC, ce qui est logique, puisqu'il a été écrit par des gens d'Apple (l'annonce est ici <https://www.ietf.org/mail-archive/web/v6ops/current/msg22455.html>, portant même sur des versions antérieures). Apple en a d'ailleurs profité pour breveter cette technologie <https://datatracker.ietf.org/ipr/3077/>. À l'inverse, un exemple récent de logiciel incapable de gérer proprement le cas d'un pair ayant plusieurs adresses IP est Mastodon (cf. bogue #3762 <https://github.com/tootsuite/mastodon/issues/3762>.)

Dans l'annexe A, vous trouverez la liste complète des importants changements depuis le RFC 6555. Le précédent RFC n'envisageait qu'un seul cas, deux adresses IP, une en v4, l'autre en v6. Notre nouveau RFC 8305 est plus riche, augmente le parallélisme, et ajoute :

— La façon de faire les requêtes DNS (pour tenir compte des serveurs bogués qui ne répondent pas aux requêtes AAAA, cf. RFC 4074),
— La gestion du cas où il y a plusieurs adresses IP de la même famille (v4 ou v6),
— La bonne façon d'utiliser les souvenirs des connexions précédentes,
— Et la méthode (dont je n'ai pas parlé ici) pour le cas des réseaux purement IPv6, mais utilisant le NAT64 du RFC 8305.