RFC 6724 : Default Address Selection for Internet Protocol version 6 (IPv6)

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 12 septembre 2012

Date de publication du RFC : Septembre 2012

https://www.bortzmeyer.org/6724.html

Lorsqu'une machine IPv6 a plusieurs adresses et qu'elle va initier une communication avec une autre machine ayant elle-même plusieurs adresses, quelle adresse choisir pour la source ? Et pour la destination ? Ce RFC donne deux algorithmes (un pour la source et un pour la destination) à suivre pour que la sélection d'adresse par défaut soit prévisible. Naturellement, il sera toujours possible, pour l'ingénieur système ou pour le programmeur, de passer outre ce choix par défaut et de choisir délibérement une autre adresse. Ce RFC remplace le RFC 3484 [1].

Alors, quel est le problème ? En IPv6, il est courant d'avoir plusieurs adresses IP pour une machine (RFC 4291). Par exemple, si un site est "*multi-homé*", les machines peuvent avoir une adresse par FAI. Ou bien il peut y avoir des tunnels, qui viennent avec leurs adresses. Toutes ces adresses peuvent avoir des caractéristiques variables : portée, statut (préférée, abandonnée, cf. RFC 4862), temporaire (pour des raisons de vie privée, cf. RFC 8981) ou pas... Voici un exemple sur une machine Debian ayant six adresses IPv6 :

```
% ip -6 addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 2605:4500:2:245b::f00/64 scope global
       valid_lft forever preferred_lft forever
    inet6 2605:4500:2:245b::bad:dcaf/64 scope global
       valid_lft forever preferred_lft forever
    inet6 2605:4500:2:245b::42/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fed9:83b3/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 fe80::5054:ff:fe9f:5bb1/64 scope link
       valid_lft forever preferred_lft forever
```

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc3484.txt

Il faudra donc un mécanisme pour décider de l'adresse **source** à utiliser.

Quant à la machine de destination, elle peut aussi avoir plusieurs adresses, par exemple plusieurs enregistrements AAAA sont dans le DNS. Il faudra donc aussi choisir l'ordre dans lequel on essaiera les adresses de **destination**. À noter que ce choix peut inclure le choix entre IPv6 et IPv4. Si les deux machines (source et destination) sont en double pile, va t-on utiliser IPv4 ou IPv6 ? (Notre RFC donne la préférence à IPv6 mais voyez plus loin pour les règles exactes.)

Notez que les deux algorithmes ne rendent donc pas le même type de résultat. Pour la destination, l'algorithme produit une **liste** d'adresses ordonnées, pouvant mêler IPv4 et IPv6. Pour la source, il produit **une** adresse, la meilleure, et forcément en IPv6 (la sélection d'une adresse source IPv4 est un problème intéressant mais non couvert par ce RFC).

Soit une application en C qui appelle la routine standard `getaddrinfo()` (RFC 3493) pour trouver les adresses IP du correspondant. Dans le cas le plus fréquent, elle va essayer toutes les adresses IP renvoyées par `getaddrinfo()`, dans l'ordre où ce dernier les a renvoyées, et l'application ne spécifiera pas son adresse source (c'est le cas simple : les variantes sont discutées plus loin). Pour mettre en œuvre ce RFC, `getaddrinfo()` va utiliser l'algorithme de sélection de la destination, pour trier la liste obtenue du DNS, et le noyau du système va utiliser l'algorithme de sélection de la source pour décider de l'adresse IP source, au moment du `connect()`.

À noter qu'essayer uniquement la première adresse IP renvoyée par `getaddrinfo()` serait une mauvaise stratégie : si elle est injoignable, aucune communication ne sera possible, alors que certaines des autres adresses marchaient peut-être. Essayer les adresses de manière strictement séquentielle n'est pas non plus optimum et le RFC 6555 conseille de le faire en partie de manière parallèle.

Si on veut résumer les deux algorithmes utilisés, on peut dire qu'ils préfèrent former des couples source/destination où les deux adresses ont la même portée, ont la portée la plus étroite possible, sont des adresses préservant la vie privée, et, en fin de compte, partagent le plus de bits de préfixe possibles.

Comme indiqué plus haut, cet algorithme pourra toujours être surmonté par un choix explicite du programmeur comme la directive `outgoing-interface` d'Unbound <`https://www.bortzmeyer.org/unbound.html`> (mal nommée puisque prenant comme paramètre une adresse IP et pas une interface) ou comme, avec OpenSSH, l'option `-b` sur la ligne de commande (ou bien `BindAddress` dans le fichier de configuration). Ou bien un choix explicite de l'administrateur système qui peut éditer une table des politiques (celle qui dit par exemple que IPv6 est préféré à IPv4, ce qu'on peut changer), modifier la préférence par défaut en faveur des adresses de préservation de la vie privée, et permettre ou pas l'addition automatique de certaines adresses à la table des politiques.

Que contient cette « table des politiques » ? La section 2.1 la détaille. Elle stocke des préfixes d'adresses IP et, comme pour une table de routage, c'est le plus spécifique (le plus long) qui gagne. Sur un système Linux, c'est en général le fichier `/etc/gai.conf` qui contient la table, fichier que l'administrateur système peut éditer selon ses goûts (section 10.3 du RFC). La table par défaut contient :

```
Prefix        Precedence Label
::1/128               50     0
::/0                  40     1
::ffff:0:0/96         35     4
2002::/16             30     2
2001::/32              5     5
fc00::/7               3    13
::/96                  1     3
fec0::/10              1    11
3ffe::/16              1    12
```

La préférence est donnée aux préfixes ayant le champ "*Precedence*" le plus élevé. Ainsi, si on a le choix entre les adresses `2001:db8:1::cafe` et `fec0::90ff:fe92:bd00`, la première a une précédence de 40 (en raison de la règle `::/0`), la seconde de 1 (en raison de la règle `fec0::/10`) : rappelez-vous que c'est le préfixe le plus spécifique qui est choisi, pas celui qui est le premier dans la table). On va donc choisir l'adresse publique, `2001:db8:1::cafe` (le préfixe `fec0::/10` a été abandonné par le RFC 3879).

Autre conséquence de cette table, IPv6 est préféré à IPv4 (représenté par `::ffff:0:0/96`, cf. RFC 4291, section 2.5.5.2). Si on veut changer cela, par exemple parce qu'on a une connectivité IPv6 vraiment trop mauvaise (voir le problème des globes oculaires <https://www.bortzmeyer.org/globes-oculaires-heureu
html> et la section 10.3.1 de notre RFC), on édite `gai.conf` ou son équivalent et on met, par exemple (syntaxe de `gai.conf`) une précédence de 100 :

```
precedence ::ffff:0:0/96  100
```

D'autres trucs Linux sont documentés dans « "*IPv6 Source Address Selection on Linux*" <http://www.davidc.net/networking/ipv6-source-address-selection-linux> ». Si on travaille sur FreeBSD, la configuration est dans `/etc/ip6addrctl.conf` et il faut donc regarder la documentation de ip6addrctl <http://www.freebsd.org/cgi/man.cgi?query=ip6addrctl&apropos=0&sektion=0&manpath=FreeBSD+8.2-RELEASE&format=html>.

Notez que cette table peut être dynamique : une implémentation a le droit d'y ajouter des préfixes en cours de route, par exemple pour ses ULA (RFC 4193).

Autre définition importante, la notion de la longueur du préfixe commun. Cette longueur est définie en comparant uniquement les 64 premiers bits de l'adresse. Ainsi, la longueur du préfixe commun à `2001:db8:1::bad` et `2001:db8:2::bad` est 32 (`2001:db8` est la seule partie commune). Mais la longueur du préfixe commun à `2001:db8:1::bad:dcaf` et `2001:db8:1::bad:cafe` n'est que de 64 et pas de 112 comme on pourrait le croire à première vue. C'est parce qu'on ignore l'"*Interface ID*" (les 64 derniers bits). C'est un des gros changements par rapport au précédent RFC, le RFC 3484, motivé par des problèmes comme une mauvaise interaction avec des systèmes de répartition de charge <https://twiki.cern.ch/twiki/bin/view/EGEE/RandomizeGetAddrInfo>..

Et les propriétés des adresses (section 3) ? On a entre autres :

— La portée : locale au lien, locale au site, ou globale. Ainsi, `fe80::35cd:efc1:e399:3aa9` a une portée inférieure à `2001:4f8:3:2bc:1::64:21`.
— Le fait d'être l'adresse « de référence » ("*home address*") ou l'adresse actuelle du mobile ("*care-of address*") lorsqu'on utilise les fonctions de mobilité du RFC 6275. En pratique, elles sont très peu déployées donc on peut ne pas faire trop attention à cette propriété.

Maintenant, les algorithmes eux-mêmes. D'abord, pour le choix de l'adresse **source** (sections 4 et 5). Il faut d'abord établir une liste d'adresses **candidates** pour la destination choisie. La méthode conseillée est que cet ensemble de candidates soit la liste des adresses attachées à l'interface réseau par laquelle les paquets vont sortir. Sur une machine Linux, si on veut écrire à `::1`, ce sera l'interface locale (`lo`) et si on veut écrire à `2001:4860:4860::8888`, ce sera la plupart du temps `eth0`. Cela permet notamment de tenir compte des filtres éventuels des routeurs (un FAI ne laissera pas passer des paquets dont l'adresse source n'est pas chez lui, cf. RFC 2827).

Une fois cette liste établie, il faut choisir l'adresse IP source (notez le singulier), parmi cet ensemble de candidates. Conceptuellement, il s'agit de trier la liste des candidates et, pour cela, il faut une fonction permettant de comparer ces adresses deux à deux. La définition de cette fonction est **le cœur de notre RFC**. Soient deux adresses SA et SB pour une destination D, ces huit règles permettent de les départager :

— Si SA = D, choisir SA (préférer une adresse source qui soit égale à la destination, pour les paquets locaux à la machine),
— Si la portée de SA est différente de celle de SB, choisir l'adresse qui a la même portée que D. En gros, pour une destination locale au lien, on préférera des adresses sources locales au lien. Exemple : destination `2001:db8:1::1`, candidates source `2001:db8:3::1` et `fe80::1`, on choisira `2001:db8:3::1`,
— Éviter les adresses marquées comme « dépassées » (RFC 4862, section 5.5.4), qu'on doit donc classer après celles qui ne sont pas dépassées,
— Si la machine utilise Mobile IP (c'est très rare), préférer l'adresse de réference ("*home address*") à l'adresse de connexion actuelle ("*care-of address*"),
— Préférer l'adresse correspondant à l'interface de sortie vers D. Notez que cette règle ne servira à rien si on a suivi le conseil de ne mettre dans l'ensemble des candidates que les adresses de l'interface de sortie,
— Utiliser l'étiquette ("*label*") attribuée dans la table des politiques. On choisit de préférence l'adresse source qui a la même étiquette que l'adresse de destination,
— Préférer les adresses temporaires du RFC 8981. Si la destination est `2001:db8:1::d5e3:0:0:1` et que les adresses candidates sont `2001:db8:1::2` (publique) et `2001:db8:1::d5e3:7953:13eb:22e8` (temporaire), on choisit `2001:db8:1::d5e3:7953:13eb:22e8`. Cette règle est un changement par rapport au prédecesseur de notre RFC, le RFC 3484. Cette préférence doit être réglable (sur Linux, c'est apparemment avec la variable sysctl `net.ipv6.conf.*.use_tempaddr` qui, mise à 1, alloue des adresses temporaires et, mise à 2, fait qu'elles sont préférées),
— Et enfin la dernière règle, mais qui sera souvent utilisée (plusieurs des règles précédentes ne s'appliquent qu'à des cas particuliers), la règle du préfixe le plus long. On choisit l'adresse source qui a le préfixe commun avec D le plus long. Si la destination est `2001:db8:1::1` et que les adresses source possibles sont `2001:db8:1::2` (48 bits communs) et `2001:db8:3::2` (32 bits communs), on sélectionne `2001:db8:1::2`.
— Si aucune des huit règles standards n'a suffi, si les deux adresses sont encore ex-aequo, le choix est spécifique à l'implémentation (par exemple, prendre la plus petite).

Une implémentation a le droit de tordre certaine de ces règles si elle dispose d'informations supplémentaires. Par exemple, si elle suit la RTT des connexions TCP, elle peut connaître les « meilleures » adresses et ne pas utiliser la huitième règle (préfixe le plus long).

Maintenant, il faut choisir l'adresse de **destination** (section 6). Cette fois, le résultat est une liste d'adresses qui seront essayées dans l'ordre. Et, contrairement au précédent, cet algorithme gère IPv6 et IPv4 ensemble (les règles de sélection de la source n'étaient valables que pour IPv6 ; par exemple la règle du plus long préfixe commun n'a guère de sens avec les courtes adresses v4). Le point de départ est la liste d'adresses récupérées en général dans le DNS. Et il y a ensuite dix règles de comparaison entre les destinations DA et DB, en tenant compte des adresses sources, S(DA) et S(DB). Beaucoup de ces règles sont proches de celles utilisées pour la sélection de la source donc je ne les réexplique pas ici :

— Éviter les destinations injoignables (la machine ne le sait pas forcément mais elle a pu acquérir cette information, par exemple via le RFC 4861, section 7.3),
— Préférer les portées identiques. Si DA et S(DA) ont une portée identique, et que ce n'est pas le cas pour DB et S(DB), choisir DA. Si les destinations connues sont `2001:db8:1::1` et `198.51.100.121`, avec comme adresses source disponibles `2001:db8:1::2`, `fe80::1` et `169.254.13.78`, alors le premier choix est `2001:db8:1::1` (source `2001:db8:1::2`) puis `198.51.100.121` (source `169.254.13.78`). Ce n'est pas une préférence pour IPv6 (elle intervient plus tard, dans la sixième règle) mais une conséquence du fait que l'adresse IPv4 `169.254.13.78` est locale au lien (RFC 3927),
— Éviter les adresses dépassées,
— Choisir les adresses de référence en cas de mobilité,
— Choisir de préférence la destination qui aura la même étiquette que l'une de ses sources possibles,
— Préferer les destinations avec la plus haute précédence (dans la table des politiques vue plus haut). Avec la table par défaut, c'est cette règle qui mènera à faire les connexions en IPv6 plutôt qu'en IPv4 (section 10.3),

— Préférer les destinations accessibles par de l'IPv6 natif plutôt que via un tunnel. Cela n'élimine pas tous les tunnels car celui-ci peut être situé plus loin, hors de la connaissance de la machine, comme avec le 6rd - RFC 5969 - de Free),
— Préférer les portées plus spécifiques, on tentera les adresses locales au lien avant les adresses publiques. Si on a le choix entre les destinations `2001:db8:1::1` ou `fe80::1`, et que les sources possibles sont `2001:db8:1::2` ou `fe80::2`, on préférera `fe80::1` (avec la source `fe80::2`),
— Utiliser le plus long préfixe commun entre la source et la destination. Si les destinations possibles sont `2001:db8:1::1` et `2001:db8:3ffe::1`, avec comme sources possibles `2001:db8:1::2`, `2001:db8:3f44::2` ou `fe80::2`, le premier choix est `2001:db8:1::1` (pour al source `2001:db8:1::2`) puis `2001:db8:3ffe::1` (pour la source `2001:db8:3f44::2`),
— Si on arrive là, ne toucher à rien : contrairement à la sélection de la source, il n'est pas nécessaire de sélectionner **une** adresse, on peut tout à fait laisser l'ordre inchangé.

Le lecteur subtil a noté que l'interaction de ces règles avec le routage IP normal n'est pas évidente. Le principe de base est qu'une machine sélectionne la route de sortie **avant** de sélectionner l'adresse source mais la section 7 de notre RFC autorise explicitement une implémentation d'IPv6 à utiliser l'adresse source pour le choix de la route.

Voilà, les deux algorithmes sont là. Mais, si vous voulez les mettre en œuvre et pas seulement comprendre comment ils marchent, la section 8 est une lecture recommandée, car elle donne de bons conseils d'implémentation. Par exemple, comme l'algorithme de sélection de la destination a besoin de connaître les adresses sources possibles, le RFC suggère à `getaddrinfo()` de demander à la couche 3, pour obtenir la liste des adresses source possibles. Il sera alors capable de renvoyer une liste de destinations déjà triée. Il peut même garder en mémoire cette liste d'adresses, pour éviter de refaire un appel système supplémentaire à chaque fois. (Le RFC dit toutefois que cette mémorisation ne devrait pas durer plus d'une seconde, pour éviter d'utiliser une information dépassée.)

Et la sécurité (section 9) ? Il n'y a que des petits détails. Par exemple, en annonçant des tas d'adresses différentes pour lui-même, un méchant peut, en examinant les adresses source de ceux qui se connectent à lui, apprendre leur politique locale (pas une grosse affaire). Autre cas amusant, en sélectionnant soigneusement les adresses source ou destination, on peut influencer le chemin pris par les paquets, ce qui peut être utile par exemple pour "*sniffer*". Mais c'est une possibilité très limitée.

Quels sont les changements de notre RFC 6724 par rapport à son prédécesseur, le RFC 3484 ? Ils sont détaillés dans l'annexe B. Les principaux (dans l'ordre à peu près décroissant d'importance) :

— Préférence désormais donnée aux adresses temporaires sur les autres, afin de mieux préserver la vie privée, désormais considérée comme plus importante (c'était déjà le cas dans plusieurs implémentations d'IPv6),
— Changement de la définition de « longueur du préfixe commun » pour se limiter aux 64 premiers bits (c'est-à-dire, excluant l'"*Interface ID*"),
— Certains changements pour mieux gérer le cas où le réseau de sortie filtre selon l'adresse source (RFC 2827 et RFC 5220),
— Ajout de la possibilité, pour une mise en œuvre d'IPv6, d'ajouter automatiquement des entrées à la table des politiques,
— Ajout d'une ligne dans la table des politiques pour les ULA du RFC 4193 (`fc00::/7`),
— Ajout d'une ligne dans la table des politiques pour les adresses du défunt 6bone (RFC 3701), avec une faible préférence pour qu'elles ne soient pas sélectionnées.

À noter que les premières versions de ce RFC étaient rédigées simplement sous forme d'une liste des différences avec le RFC 3484. C'était illisible.

Enfin, une anecdote, le prédécesseur, le RFC 3484, avait été accusé d'un problème sérieux <`http://drplokta.livejournal.com/109267.html`>. Mais c'est contestable, les responsabilités sont partagées : l'algorithme du RFC 3484 (qui ne concerne qu'IPv6) est certes discutable, l'implémentation de

Vista n'est pas géniale (extension du RFC 3484 à IPv4, où les préfixes sont bien plus courts), la configuration de l'auteur de l'article est mauvaise (il dépend de l'ordre des enregistrements, qui n'est pas garanti par le DNS, c'est la même erreur qu'un programmeur C qui dépendrait de la taille d'un `int`) et, pour couronner le tout, le NAT qui fait qu'une machine ne connait pas son adresse publique.

Une intéressante discussion de certaines objections à ce RFC a eu lieu sur SeenThis `<http://seenthis.net/messages/86502>`.