RFC 6147 : DNS64: DNS extensions for Network Address Translation from IPv6 Clients to IPv4 Servers

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 avril 2011

Date de publication du RFC : Avril 2011

https://www.bortzmeyer.org/6147.html

Ce nouveau RFC est un composant indispensable de la nouvelle technique de traduction entre des machines IPv4 et IPv6 (dont une description générale se trouve dans le RFC 6144 [1]). Les machines purement IPv6 ne pouvant pas initier une communication avec des machines n'ayant qu'une adresse IPv4, il est nécessaire de les « tromper » en fabriquant dynamiquement des adresses IPv6. Les machines IPv6 tenteront alors de s'y connecter, leurs requêtes seront reçues par le traducteur d'adresses, qui pourra alors les convertir en IPv4 (cf. RFC 7915 et RFC 6146).

DNS64 n'est donc pas utilisable seul. Il doit se combiner à une machine, typiquement un routeur, qui fait la traduction d'adresses, dite NAT64 <https://www.bortzmeyer.org/nats.html>. La configuration typique des utilisateurs de DNS64 sera un réseau local purement IPv6 (puisque le stock d'adresses IPv4 est épuisé <https://www.bortzmeyer.org/epuisement-adresses-ipv4.html>), dont le routeur fait de la traduction NAT64, et qui est muni d'un résolveur DNS capable de fabriquer des adresses IPv6 à partir des adresses v4 des machines externes. En effet, le but du système NAT64 est de ne pas avoir à modifier les machines, seulement les équipements d'infrastructure (routeur et résolveur DNS). Les machines purement v6, ignorantes de la traduction qui se fait entre leur réseau et le monde extérieur, ne pourraient pas se connecter si elle ne connaissaient de leur pair qu'une adresse IPv4. NAT64 et son indispensable compagnon DNS64 rejoignent donc la boîte à outils des techniques permettant de faire coexister IPv4 et IPv6 (cf. RFC 6144).

Comment est-ce que le serveur DNS64 synthétise cette adresse IPv6 ? Supposons qu'une machine IPv6 du réseau local veuille se connecter à www.example.com et que celui-ci n'ait qu'une adresse v4, mettons 203.0.113.18. Le navigateur Web tournant sur le client va faire une requête DNS de type

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6144.txt

AAAA (adresse IPv6 ; puisqu'il n'a pas d'IPv4, les requêtes de type A n'auraient aucun intérêt). Le serveur DNS la reçoit, la relaie et récupère une réponse vide (le nom existe mais n'a pas d'enregistrement AAAA). Il tente alors une requête A, reçoit `203.0.113.18` en réponse et génère l'adresse v6 de manière purement algorithmique, sans état, en utilisant l'adresse v4 et les paramètres avec lesquels le serveur a été configuré (typiquement, un préfixe v6). Rappelez-vous donc bien que le serveur DNS64 et le traducteur doivent être configurés de manière cohérente (dans la plupart des cas, ils seront sans doute dans la même boîte mais, conceptuellement, ce sont deux services séparés.)

Pour le lecteur pressé qui veut juste une introduction à la norme, la section 2 est la meilleure lecture. Voici un exemple où une machine purement IPv6 tente de contacter Twitter, service purement IPv4 :

```
% telnet twitter.com 80
Trying 64:ff9b::80f2:f0f4...
Connected to twitter.com.
Escape character is '^]'.
```

C'est le serveur DNS64 qui a généré l'adresse `64:ff9b::80f2:f0f4` et c'est le traducteur qui changera les paquets IPv6 en IPv4 à l'aller et en sens inverse au retour. Twitter et le client purement IPv6 n'y verront que du feu. Pour le préfixe à utiliser, le serveur DNS64 et le traducteur ont le choix (cf. RFC 6052) entre un préfixe bien connu, réservé à la traduction d'adresse (celui utilisé dans l'exemple, qui avait été testé lors d'une réunion IETF à Pékin) ou bien un préfixe appartenant à l'opérateur réseau (NSP, pour "*Network-Specific Prefix*"). La seule contrainte est qu'il doit être configuré de manière **identique** dans le serveur DNS64 et dans le traducteur.

La norme elle-même fait l'objet de la section 5. Elle décrit avec précision le fonctionnement que j'ai résumé plus haut. Quelques petits points intéressants :

— On ne synthétise d'enregistrements que s'il n'y avait pas déjà un AAAA (section 5.1.1).
— Les erreurs (autres que NXDOMAIN - "*No Such Domain*") doivent être traitées comme si le AAAA étai absent (section 5.1.2). Donc, par exemple, SERVFAIL ("*Server Failure*") doit être traité comme un NOERROR,ANSWER=0, pour tenir compte des serveurs assez pourris pour ne pas répondre correctement aux requêtes AAAA (cf. RFC 4074). C'est une violation des normes du DNS, rendue nécessaire par le nombre de serveurs DNS (ou de "*middlewares*" placés devant eux) programmés avec les pieds.
— La règle précédente, interdisant la synthèse d'enregistrements AAAA s'il en existe déjà, a quelques exceptions (section 5.1.4). Ainsi, les adresses IPv6 non utilisables globalement comme celles en `::ffff:0:0/96` doivent être ignorés et le AAAA de synthèse produit.
— Si le AAAA n'existe pas ou est inutilisable (section 5.1.6), le serveur DNS64 demande un enregistrement A (adresse IPv4). S'il n'en trouve pas, il répond au client « Désolé, aucune réponse ». S'il en trouve un, il fabrique l'enregistrement AAAA correspondant (typiquement en concaténant le préfixe IPv6 avec l'adresse IPv4) et le renvoie (la réponse de type A, elle, n'est pas transmise). La section 5.1.7 donne les détails de cette synthèse.
— Les deux requêtes, pour un enregistrement de type AAAA et pour un enregistrement de type A, peuvent être faites en parallèle (section 5.1.8). Certes, ce sera un gaspillage si l'enregistrement AAAA existe (puisque le A sera alors inutile) mais cela permettra de diminuer le temps total d'attente si le AAAA n'existe pas.

J'ai dit plus haut que l'adresse IPv6 de synthèse était fabriquée par concaténation du préfixe IPv6 (configuré à la main dans le serveur DNS64) et de l'adresse IPv4. Mais, en fait, le RFC n'impose pas l'utilisation d'un tel algorithme. La méthode utilisée n'a pas besoin d'être normalisée (elle n'est pas vue à l'extérieur) et doit seulement être la même sur le serveur DNS64 et sur le traducteur. La section 5.2 précise plus rigoureusement les exigences auxsquelles doit obéir cet algorithme. Le point le plus important est qu'il doit être **réversible** : en échange de l'adresse IPv6 de synthèse, le traducteur doit pouvoir retrouver l'adresse IPv4 originelle. Pour faciliter la vie des administrateurs réseau, un algorithme doit

être obligatoirement présent dans les programmes (mais ce n'est pas forcément celui que l'administrateur choisira), celui décrit en section 2 du RFC 6052.

Et que doit faire le serveur DNS64 s'il reçoit des requêtes d'un autre type que A ou AAAA ? La section 5.3 couvre ce cas. En gros, il doit les traiter normalement **sauf** les requêtes de type PTR (résolution d'une adresse en nom). Dans ce cas, le serveur doit extraire l'adresse du nom en `ip6.arpa` (section 2.5 du RFC 3596), regarder si cette adresse correspond à un des préfixes qu'il gère et, si oui, le serveur DNS64 a deux possibilités :

— Répondre immédiatement, avec autorité, en utilisant un nom adapté (qui peut être le même pour toutes les adresses, mettons `locally-generated-ipv6-address-with-dns64.example.net`). L'avantage est que les clients DNS verront bien que l'adresse avait été synthétisée, l'inconvénient est qu'on n'utilisera pas les informations qui pouvaient se trouver dans le DNS,
— Synthétiser les enregistrements PTR à partir de données trouvées dans le DNS, comme il synthétise des enregistrements AAAA à partir des A récupérés dans le DNS. Dans ce cas, le serveur DNS64 traduit l'adresse IPv6 en IPv4, crée un nom dans `in-addr.arpa`, fait une requête PTR et renvoie le résultat au client.

La section 6, sur le déploiement, est consacrée aux problèmes pratiques qui pourront survenir lorsqu'on utilisera réellement DNS64. Par exemple, si une machine est "*multi-homée*", il est possible que seulement certaines de ses connexions à l'Internet passent par un traducteur v4-v6. Il faudrait donc idéalement interroger des résolveurs DNS différents selon l'interface de sortie, et considérer les résultats comme locaux à l'interface.

Autre problème pratique, le cas d'une machine à double-pile qui utiliserait DNS64, passant ainsi par le traducteur même lorsque ce n'est pas nécessaire (section 6.3.2 et 6.3.3). Comme la configuration par défaut des systèmes d'exploitation est de préférer IPv6 à IPv4, le cas risque de se produire souvent. Il n'existe pas de moyen simple de le détecter puisque, pour le client, un serveur DNS64 semble un serveur comme un autre. Une machine double-pile (IPv4 et IPv6) ne devrait donc pas utiliser de résolveur DNS64. Mais il peut y avoir des cas où c'est nécessaire, par exemple si la machine double-pile n'a une connectivité IPv4 que partielle. Dans ces conditions, le serveur DNS64 doit exclure les préfixes IPv4 locaux de la synthèse. On peut donc prévoir quelques problèmes de débogage amusants...

La section 7 décrit des exemples de déploiement, en utilisant les scénarios du RFC 6144. Par exemple, la section 7.1 cite le cas qui sera sans doute le plus fréquent au début, une machine H1 réseau purement IPv6 (parce que arrivé après l'épuisement des adresses IPv4 <https://www.bortzmeyer.org/epuisement-adresses-ipv4.html>) qui essaie de se connecter à un serveur H2 situé dans l'Internet v4 traditionnel. Le traducteur T et le serveur DNS sont configurés avec le préfixe bien connu `64:ff9b::/96`. H1 va commencer par faire une requête AAAA pour `h2.example.com`. Le résolveur DNS ne trouve pas de AAAA pour ce nom, mais il récupère un enregistrement de type A, `192.0.2.1`. Il fabrique alors le AAAA `64:ff9b::c000:201` (qui peut aussi légitimement s'écrire `64:ff9b::192.0.2.1`). H1 va alors envoyer un paquet TCP `SYN` à `64:ff9b::c000:201`. Le routeur/traducteur le voit passer, note le préfixe NAT64 avec lequel il a été configuré et le traduit en un paquet TCP/IPv4.

Autre exemple, en section 7.3. C'est à peu près l'inverse : une machine dans un Internet passé presque entièrement en IPv6 essaie de se connecter à un des derniers serveurs v4 existants. À noter que DNS64 n'est pas indispensable dans ce cas : on peut simplement affecter algorithmiquement une adresse IPv6 à toutes les adresses IPv4, les publier sur des serveurs DNS normaux, et donc n'utiliser que le traducteur. Toutefois, cette section cite quelques cas où DNS64 peut être utile, par exemple si les adresses IPv4 sont affectées dynamiquement.

Fondamentalement, un serveur DNS64 est un menteur : il fabrique des réponses qui n'existent pas (Twitter, utilisé dans l'exemple plus haut, n'a pas d'adresse IPv6 à ce jour...). Il a donc des problèmes avec DNSSEC, qui a justement été conçu pour détecter les mensonges. Toute la section 3 est consacrée

à discuter ce problème. Bien sûr, le résolveur DNS64 ment pour la bonne cause. Mais DNSSEC est au delà du bien et du mal, il se moque des motivations, il ne regarde que le résultat. Plusieurs cas peuvent se poser (le RFC en identifie sept !) selon les options indiquées dans la requête DNS et selon la configuration DNSSEC utilisée. Rappelons que les deux options importantes pour DNSSEC dans une requête DNS sont le bit DO ("*DNSSEC OK*", RFC 4035, section 3.2.1) qui indique si le client DNS comprend les enregistrements DNSSEC et le bit CD ("*Checking Disabled*", RFC 4035, section 3.2.2) qui indique au résolveur que le client fera la validation lui-même et qu'il faut lui envoyer toute l'information, même si elle s'avère fausse.

Les différents cas sont donc (je ne les ai pas tous mentionnés, voir le RFC pour les détails) :

— Un serveur DNS64 reçoit une requête où DO est à zéro (ce que font aujourd'hui la plupart des bibliothèques de résolution de noms, les "*stub resolvers*"). Aucun problème dans ce cas, le client ne comprend pas DNSSEC, le serveur DNS64 peut inventer tous les mensonges qu'il veut.
— Un serveur DNS64 avec gestion de DNSSEC reçoit une requête avec DO et CD mis à zéro (c'est typiquement le cas - rare - d'un client qui veut valider lui-même). Dans ce cas, il doit passer au client toute l'information obtenue. DNS64 ne marchera **pas** ici, le client n'acceptera pas le AAAA synthétisé comme étant valide. La seule solution est que le client mette en œuvre DNS64 lui-même.
— Un serveur DNS64 qui valide avec DNSSEC reçoit une requête avec DO à un et CD à zéro (ce sera sans doute le cas le plus courant dans le futur). Aucun problème, le serveur DNS64 valide les données récupérées et répond SERVFAIL ("*Server Failure*") si cette validation échoue. Le AAAA synthétisé n'est pas validé avec DNSSEC mais le client ne le saura pas. Si le bit DO était mis à un, le serveur DNS64 a juste à mettre le bit AD ("*Authentic Data*") à un dans sa réponse (un mensonge pour la bonne cause ; après tout, si le client DNS n'a pas mis CD à un, c'est qu'il fait complètement confiance au résolveur).

La section 5.5 donne les détails sur le fonctionnement de DNSSEC dans DNS64. La section 6.2 rappelle les problèmes pratiques de déploiement de DNS64 si on utilise déjà DNSSEC.

DNS64 pose t-il des problèmes de sécurité ? Pas beaucoup, dit la section 8. Comme indiqué (paragraphes précédents), il peut interférer avec DNSSEC. Et, surtout, il impose une coordination entre le serveur DNS64 et le traducteur. Si un attaquant peut modifier la valeur du préfixe du serveur DNS64, il peut envoyer les paquets où il veut. Donc, attention, la sécurité du serveur DNS64 est exactement aussi importante que celle du traducteur.

Enfin, l'annexe A explique des motivations pour générer un AAAA alors qu'il existe déjà. Il se peut que la meilleure route passe plutôt par le traducteur et donc, dans certains cas, un serveur DNS64 peut offrir à son administrateur la possibilité de permettre la synthèse d'enregistrements AAAA même quand un vrai existe.

BIND met en œuvre DNS64 depuis les versions 9.7.3 et 9.8.0. Voici un exemple de configuration de DNS64 avec une 9.8.0 :

```
acl me {
    ::1/128; 127.0.0.0/8;
};
acl rfc1918 {
    10/8; 192.168/16; 172.16/12;
};

options {
       ...
       dns64 64:ff9b::/96 { // The well-known prefix
              clients { me; };
              mapped { !rfc1918; any; }; // Never synthetize AAAA records
                     // for private addresses
```

```
        exclude { 64:ff9b::/96; ::ffff:0000:0000/96; }; // If the retrieved
        // AAAA record is in that range, do not send it back to the user,
        // synthetize a AAAA.
        //
        // suffix ::42; // Trailing bits. Require some free space (here,
        // 96 + 32 bits leave no room for trailing bits).
    };
```

Le premier préfixe est celui des AAAA synthétisés (ici, le préfixe bien connu). La directive `clients` réduit le service DNS64 à certaines machines seulement (celles qui sont connectées via un traducteur NAT64, relisez les sections 6.3.2 et 6.3.3 du RFC). La directive `mapped` met en œuvre la règle d'exclusion des préfixes IPv4 locaux (ici, ceux du RFC 1918). La directive `exclude` implémente la règle d'exclusion des préfixes IPv6 qui ne devraient même pas être mis dans le DNS global. S'il existe un AAAA, mais qu'il pointe vers un des ces préfixes, il sera remplacé par un AAAA synthétique. Enfin, la directive `suffix` permet d'ajouter quelques bits à la fin du AAAA de synthèse (mais, ici, avec 96 bits pour le préfixe IPv6 et 32 pour l'adresse IPv4, il n'y a plus de place).

Voici l'effet du serveur ayant la configuration ci-dessus (il écoute sur le port 9053) :

```
[Adresse IPv6 déjà dans le DNS global.]
% dig +short -p 9053 @::1 AAAA www.ietf.org
2001:1890:1112:1::20

[Pas d'adresse IPv6 dans le DNS global mais trois adresses IPv4.]
% dig +short -p 9053 @::1 AAAA facebook.com
64:ff9b::453f:bd0b
64:ff9b::453f:bd10
64:ff9b::453f:b50c

[Les requêtes PTR marchent bien, BIND synthétise un CNAME, une des
deux techniques permises par le RFC.]
% dig  -p 9053 @::1 -x 64:ff9b::453f:bd10
...
;; QUESTION SECTION:
;0.1.d.b.f.3.5.4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.b.9.f.f.4.6.0.0.ip6.arpa. IN PTR

;; ANSWER SECTION:
0.1.d.b.f.3.5.4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.b.9.f.f.4.6.0.0.ip6.arpa. 600 IN CNAME 16.189.63.69.in-addr.arpa
16.189.63.69.in-addr.arpa. 7172 IN      PTR     www-11-01-ash2.facebook.com.
...

[Ici, le nom a une adresse IPv4 mais privée. L'ACL 'rfc1918' va donc
la bloquer.]
% dig +short -p 9053 @::1 A ciel.administratif.prive.nic.fr
10.1.84.200

% dig  -p 9053 @::1 AAAA ciel.administratif.prive.nic.fr
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 0
                          ^^^^^^^^^
```

Pour comparer, voici une configuration BIND avec un préfixe du FAI (NSP pour "*Network-Specific Prefix*"). Plus court que le précédent, il dégage de la place pour le suffixe :

```
    dns64 2001:db8:666::/48 { // A Network-Specific Prefix (NSP).
        clients { me; };
        suffix ::42; // Trailing bits.
    };
```

Et voici les résultats :

```
% dig +short -p 9053 @::1 AAAA facebook.com
2001:db8:666:453f:bd:1000:0:42
2001:db8:666:453f:b5:c00:0:42
2001:db8:666:453f:bd:b00:0:42
```

Si vous voulez manipuler les adresses vous-même, voici un joli script awk dû à gbfo <http://twitter.com/gbfo>, pour traduire les adresses IPv6 en la v4 correspondant :

```
% echo 64:ff9b::453f:bd0b | \
        awk -F: '{h=strtonum("0x" $(NF-1));l=strtonum("0x" $NF);printf("%d.%d.%d.%d\n",h/256,h%256,l/256,l%2
69.63.189.11
```

Question mises en œuvre de DNS64, outre le cas de BIND, déjà présenté, il existait une rustine pour Unbound, chez Viagénie <http://ecdysis.viagenie.ca/download.html> (avec un très ennuyeux système de formulaire à remplir pour l'obtenir). Cette rustine porte sur une version ancienne du protocole, mais Unbound a désormais DNS64 en série. Il y a aussi au même endroit <http://ecdysis.viagenie.ca/download.html> un serveur DNS64 autonome, écrit en Perl. Ce dernier est très court et peut donc être un bon moyen de comprendre le protocole, pour ceux qui aiment lire le code source. Enfin, Cisco a désormais DNS64 <http://gblogs.cisco.com/fr-ipv6/2012/01/26/cnr-supporte-dns64/> sur ses produits.

Voici un exemple de configuration Unbound pour un serveur DNS64 :

```
server:
   module-config: "dns64 validator iterator"
   dns64-prefix: 64:ff9b::0/96
```

Et mon opinion personnelle sur DNS64 ? Je dirais que, certes, c'est un "*hack*", mais qu'on a vu pire et que ça traite un vrai problème. DNS64 semble bien pensé, rien n'est oublié et le RFC parle bien de tous les problèmes, notamment avec DNSSEC.

Pour un récit très détaillé, avec plein de technique, d'un test de NAT64 avec DNS64, voir l'article de Jérôme Durand « J'ai testé pour vous : Stateful NAT64 avec DNS64 <http://gblogs.cisco.com/fr-ipv6/2011/09/26/jai-teste-pour-vous-stateful-nat64-avec-dns64/> ».

Notez qu'il existe des résolveurs DNS publics ayant DNS64 (évidemment uniquement avec le préfixe bien connu), comme celui de Hurricane Electric, nat64.he.net / 2001:470:64::1 ou celui de Google, google-public-dns64-b.google.com / 2001:4860:4860::6464.