RFC 6146 : Stateful NAT64: Network Address and Protocol 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/6146.html

Dans la grande série des techniques de coexistence entre IPv4 et IPv6, voici une normalisation de **NAT64**, une technique de traduction entre IPv6 et IPv4, permettant à une machine purement IPv6 de se connecter à une machine purement IPv4. Ce RFC fait partie de la série introduite par le document d'architecture générale RFC 6144 [1].

Aujourd'hui, l'état de l'Internet est que la majorité des serveurs sont accessibles uniquement en IPv4. Or, les dernières adresses IPv4 disponibles à l'IANA ont été distribuées le 3 février 2011 <https://www.bortzmeyer.org/epuisement-adresses-ipv4.html>. Demain, les nouvelles machines recevront donc de plus en plus uniquement des adresses IPv6. Comment se connecteront-elles à ces serveurs purement v4 ? Une des solutions est de traduire automatiquement les paquets IPv6 du client en IPv4 pour les envoyer au serveur, et de faire la traduction inverse au retour. Cela se nomme NAT64 et notre RFC normalise la version « à état » de cette méthode. « À état » car le traducteur garde un souvenir des flux de données en cours et peut donc, pour la traduction, tenir compte d'un état, des paquets précédents (la version sans état est dans le RFC 7915).

Notez le titre de ce RFC : ses ambitions sont limitées, il ne s'agit pas, comme l'essayait le défunt NAT-PT (RFC 2766) de fournir une solution générale à tous les problèmes de communication v4-v6 mais seulement de permettre à des **clients** IPv6 de se connecter à des **serveurs** IPv4.

NAT64 ressemble beaucoup à l'actuel NAT44 <https://www.bortzmeyer.org/nats.html> (celui que tout le monde est obligé d'utiliser à l'hôtel, chez lui s'il a plusieurs machines, cf. RFC 3022) dans la mesure où il permet :

— Aucune modification au client ou au serveur, tout est fait dans le routeur du réseau IPv6-pur,

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

— Les communications client-serveur, comme le titre de notre RFC le précise,
— Certaines communications pair-à-pair, en utilisant des techniques comme ICE (RFC 5245),
— Des communications lancées depuis l'extérieur, si on a configuré statiquement le traducteur pour reconnaître certaines adresses.

Il s'en distingue par le fait qu'on ne se contente pas de traduire les adresses, il faut traduire tout l'en-tête du paquet, et par le fait que, pour que les applications tournant sur la machine purement IPv6 trouvent une adresse IPv6, un serveur DNS spécial, DNS64 (RFC 6147) est nécessaire. Vu la nécessité de faire plus de travail qu'en NAT44, la spécification ne fonctionne que pour certains protocoles des couches supérieures, pour l'instant uniquement ICMP, UDP et TCP. Ne comptez donc pas faire, par exemple, du SCTP.

Notre RFC s'appuie sur plusieurs autres documents, celui d'architecture (RFC 6144), celui sur la traduction des adresses (RFC 6052), et celui sur l'algorithme de traduction (RFC 7915).

Les équipements qui font faire le travail sont donc :

— Le traducteur, qui sera typiquement embarqué dans le routeur (CPE, "*box*", ou bien plus loin dans le réseau de l'opérateur),
— Le résolveur DNS64, qui pourra être inclus dans le même routeur ou bien fourni séparement.

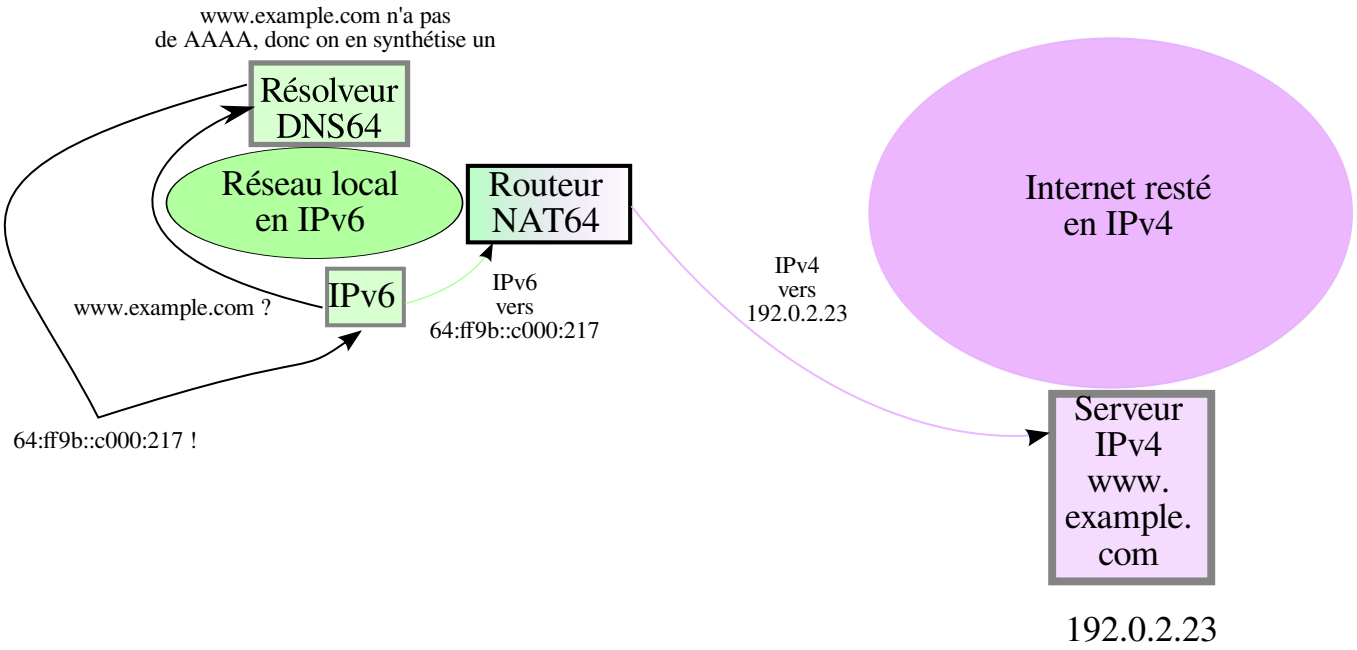
Normalement, les traducteurs qui suivront ce RFC seront conformes aux recommandations qu'avait édicté le groupe de travail Behave <https://www.bortzmeyer.org/behave-wg.html>, dans sa première version, à savoir les RFC 4787, RFC 5382, RFC 5508... et ils permettront les techniques de passage au travers des NAT comme ICE (RFC 5245).

La norme elle-même commence en section 3 mais une gentille introduction, utilisable si vous ne connaissez pas grand'chose aux NAT, figure en section 1.2. Essayons de la résumer sommairement en prenant pas mal de raccourcis : le traducteur a donc deux interfaces, une vers le réseau IPv6-pur et une vers un réseau IPv4 (et peut-être aussi IPv6 mais, dans ce cas, pas besoin de traduction). Le traducteur doit donc avoir une adresse IPv4 publique. Lorsqu'il reçoit un paquet IPv6 dont l'adresse de destination indique qu'il doit être traduit, il le transforme en IPv4 (l'adresse IPv4 est encapsulée dans l'adresse IPv6), et l'envoie par l'interface IPv4. Au retour, c'est un peu plus dur car l'adresse IPv4 de destination est celle du traducteur et, pour retrouver l'adresse IPv6 du vrai destinataire, il doit consulter une table où il a stocké les clients IPv6 qui parlaient à l'extérieur. Comme en NAT44, le numéro de port est utilisé pour reconnaitre les clients locaux. La combinaison d'une adresse IP et d'un port est appelée « adresse de transport ». On le voit, cela ne marche bien que si c'est le réseau IPv6 qui initie la communication (mais diverses astuces permettent des communications un peu plus pair-à-pair).

Le NAT64 devrait donc être appelé NAPT ("*Network Address AND PORT Translation*") car, dans l'écrasante majorité des cas, le traducteur n'aura pas assez d'adresses IPv4 pour tous les clients IPv6 qu'il sert (si on avait assez d'adresses v4, il n'y aurait guère de raison de migrer vers v6).

Comment le client IPv6 a-t-il trouvé l'adresse IPv6 de destination, puisque le serveur n'a qu'une adresse IPv4 ? C'est le rôle du serveur DNS64 qui ment (pour la bonne cause) en fabriquant un enregistrement DNS de type AAAA (adresse IPv6) à partir de l'enregistrement réel A (adresse IPv4). Les étapes complètes effectuées par le client sont décrites en section 1.2.2.

On a vu que cette opération nécessitait de réserver une partie de l'espace d'adressage IPv6 pour représenter les adresses IPv4. Cette partie peut être localement définie (après tout, seuls le traducteur et le résolveur DNS64 auront besoin de la connaître) ou bien on peut utiliser le préfixe standard `64:FF9B::/96`. Ici, on voit un serveur DNS64 et un traducteur coopérer pour permettre à une machine purement IPv6 de communiquer avec `www.example.com`, qui n'a qu'une adresse IPv4 : .



Du fait que le traducteur a un état, tous les paquets IPv4 venus de l'extérieur ne sont pas acceptés. Seuls peuvent être transmis ceux qui correspondent à une correspondance existante, en suivant les règles des RFC 4787 et RFC 5382 (*"Endpoint-Independent Filtering"* et *"Address-Dependent Filtering"* ; suivre ces règles est important pour que les applications pair-à-pair fonctionnent).

Voilà pour l'introduction. Maintenant, pour lire la spécification elle-même, il faut d'abord bien apprendre la terminologie, en section 2. Quelques termes qui ne sont sans doute pas familiers à tous :

— Un **quintuplet** est l'identificateur d'une session TCP ou UDP. Il est composé de {adresse IP source, port source, adresse IP destination, port destination, protocole de couche 4}.
— La **BIB** (*"Binding Information Base"*) est la liste des communications actuellement en cours, pour un protocole de transport donné. Une liaison (*"binding"*) est faite entre une adresse de transport IPv4 et une IPv6, afin de déterminer à qui « appartient » un paquet entrant dans le traducteur. Les liaisons peuvent être établies automatiquement, lorsque le premier paquet passe, ou bien manuellement. Voir la section 3.1.
— Un virage en épingle à cheveux désigne le cas où deux machines situées du même côté du traducteur communiquent via le traducteur, qui doit alors renvoyer le paquet par là d'où il est venu (cf. section 3.8).
— Une **adresse de transport** est le doublet formé d'une adresse IP et d'un port. Comme le traducteur aura moins d'adresses IPv4 que de clients, l'ajout du port est nécessaire pour différencier les sessions, lorsqu'un paquet IPv4 arrive.

Vous avez bien retenu tout le vocabulaire ? Alors, vous pouvez lire la norme elle-même, à partir de la section 3. Quelques points importants à retenir :

— NAT64 doit garder une table des **sessions** TCP et UDP en cours, pour pouvoir déterminer leur fin (et donc la suppression des liaisons associées).
— Chaque entrée de la BIB peut être associée à plusieurs sessions et il ne faut donc la supprimer que lorsque la dernière session est finie.
— La fragmentation pose des défis particuliers. Un traducteur NAT64 conforme à la norme doit gérer les fragments, même s'ils arrivent dans le désordre. Il doit donc faire du réassemblage (en limitant toutefois les ressources allouées à cette tâche, pour se protéger contre les attaques DoS ; voir les RFC 1858, RFC 3128 et RFC 4963). Le traducteur a aussi le droit d'ajouter quelques contraintes comme le fait que tout l'en-tête TCP doit être dans le premier fragment.

— Comme le NAT64 avec état n'est défini que pour TCP, UDP et ICMP, le traducteur doit jeter les paquets des autres protocoles (en prévenant la source, via un paquet ICMP).
— Le traducteur ayant une connaissance des sessions actives, il peut jouer un rôle de filtre en bloquant les paquets ne correspondant pas à une telle session.
— Déterminer qu'une session est finie n'est pas forcément évident. Ainsi, si la fin d'une session TCP est facile à observer (échange des paquets `FIN` par les deux parties), que faire pour une « session » UDP puisqu'il n'y a pas de connexion et donc pas de fin explicite ? La solution est d'utiliser une minuterie, remise à zéro par chaque paquet, et qui se déclenche après un certain temps d'inactivité, coupant la session UDP.
— Pour TCP, la section 3.5.2 décrit en détail les événements de la machine à états de TCP que le traducteur doit observer pour découvrir début et fin de la session. Elle est très longue et détaillée, j'espère que les programmeurs des routeurs bas de gamme à 60 dollars prendront le temps de la lire !
— Les règles de conversion des adresses IPv4 en IPv6 et réciproquement sont celles du RFC 6052.

Vous avez noté qu'à plusieurs reprises, j'ai utilisé des termes vagues comme « un certain temps ». Les valeurs numériques recommandées pour les diverses constantes sont en section 4. Ainsi, avant de fermer une session UDP, le traducteur doit attendre (par défaut) au moins cinq minutes (cf. RFC 4787), le traducteur doit attendre les fragments manquants au moins deux secondes, etc.

Cette technique du NAT64 améliore-t-elle ou aggrave-t-elle les problèmes de sécurité ? D'abord, cette section note que toute solution de sécurité de bout en bout qui protège l'en-tête IP contre les modifications (par exemple l'AH d'IPsec) va être cassée par NAT64, qui modifie précisément l'en-tête. Si le NAT64 est obligatoire, IPsec peut, par exemple, utiliser l'encapsulation UDP (RFC 3948). La section 5 insiste d'autre part sur le fait qu'un traducteur n'est pas un pare-feu et que le filtrage réalisé par effet de bord de la gestion d'état n'est pas forcément adapté à la sécurité <`https://www.bortzmeyer.org/nat-et-securite.html`>.

D'autre part, NAT64 a ses propres vulnérabilités. Les principales sont le risque d'attaques par déni de service. Par exemple, un attaquant situé à l'intérieur (côté IPv6) peut utiliser toutes les adresses (en général une seule) et ports disponibles et empêcher ainsi l'accès des autres utilisateurs. Une autre attaque possible est d'envoyer régulièrement des paquets de façon à maintenir une liaison active et d'empêcher le traducteur de relâcher des ressources. Une protection possible serait de ne remettre à zéro la minuterie que si le paquet qui passe vient de l'intérieur (supposé plus fiable).

Quelles implémentations existent aujourd'hui ? Viagénie <`http://www.viagenie.ca/`> en à une en logiciel libre, voir <`http://ecdysis.viagenie.ca/`>. Le terme de "*ecdysis*" désigne la mue des arthropodes puisque NAT64 permet à IP de muer vers un nouveau protocole... Voici un exemple d'installation et d'usage : c'est un module noyau, donc il faut installer les paquetages qui permettent la compilation de modules noyau. Sur une Debian version « squeeze » avec le noyau Linux 2.6.32-5 :

```
% sudo aptitude install linux-headers-2.6.32-5-686 linux-kbuild-2.6.32
...

% make
/bin/sh: [[: not found
make -C /lib/modules/2.6.32-5-686/build M=/home/stephane/tmp/ecdysis-nf-nat64-20101117 modules
make[1]: Entering directory '/usr/src/linux-headers-2.6.32-5-686'
  CC [M]  /home/stephane/tmp/ecdysis-nf-nat64-20101117/nf_nat64_main.o
  CC [M]  /home/stephane/tmp/ecdysis-nf-nat64-20101117/nf_nat64_session.o
  CC [M]  /home/stephane/tmp/ecdysis-nf-nat64-20101117/nf_nat64_config.o
  LD [M]  /home/stephane/tmp/ecdysis-nf-nat64-20101117/nf_nat64.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/stephane/tmp/ecdysis-nf-nat64-20101117/nf_nat64.mod.o
  LD [M]  /home/stephane/tmp/ecdysis-nf-nat64-20101117/nf_nat64.ko
make[1]: Leaving directory '/usr/src/linux-headers-2.6.32-5-686'
```

```
% sudo make install
...

# On peut éditer le script avant pour changer les paramètres...
% sudo ./nat64-config.sh
```

Ensuite, on peut vérifier que tout s'est bien passé en regardant la table de routage :

```
% netstat -r -n  -Ainet6
Kernel IPv6 routing table
Destination                    Next Hop                   Flag Met Ref Use If
64:ff9b::/96                   ::                         U    1024 0     0 nat64
...
```

Parfait, les paquets pour le préfixe NAT64 sont bien routés à part. Tentons notre chance :

```
% telnet 64:ff9b::453f:bd0b
Trying 64:ff9b::453f:bd0b...
```

Et on voit bien les paquets IPv6 sur l'interface nat64 :

```
21:18:58.910669 IP6 2a01:e35:8bd9:8bb0:304b:5a4b:3b8c:7b1c.37033 > \
         64:ff9b::453f:bd0b.23: Flags [S], seq 3580860303, win 5760, \
           options [mss 1440,sackOK,TS val 23731660 ecr 0,nop,wscale 6], length 0
```

Et les IPv4 sur l'interface normale :

```
21:19:09.263613 IP 192.168.2.1.37038 > \
         69.63.189.11.23: Flags [S], seq 4043093045, win 5760, \
           options [mss 1440,sackOK,TS val 23734248 ecr 0,nop,wscale 6], length 0
```

Un déploiement expérimental de NAT64 est celui du réseau de la recherche lituanien <https://labs.ripe.net/Members/raimis/experimental-nat64-dns64-service/view>. Voir leur site Web <http://ipv6.lt/nat64_en.php>.

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://ipv6blog.cisco.fr/2011/09/26/jai-teste-pour-vous-stateful-nat64-avec-dns64/> ».