Julien VAUBOURG
✉ julien@vaubourg.com
Stagiaire 2A (TELECOM Nancy)

Lothaire Yarding

Yet Another Reference for Delivering IPv6 to the Next Generation

Le 17 septembre 2012

UNIVERSITÉ DE LORRAINE | Lothaire

Table des matières

Préambule

« *Trainee. New recruit. It was either that or hairdressing.* » [1] - Tenth Doctor

Ce document a été réalisé dans le cadre d'un stage ingénieur TELECOM Nancy de seconde année, sur une durée de trois mois.

Vous y trouverez un état de l'art probablement non-exhaustif mais suffisamment complet pour avoir une bonne vue d'ensemble de ce qui est possible actuellement en IPv6. À partir de celui-ci, cette documentation tentera entre autres de répondre aux questions :

- *Est-il possible de migrer son réseau interne en IPv6 uniquement, tout en continuant à bénéficier des services restés en IPv4 ?*
- *Est-il possible de se passer totalement de l'IPv4 ?*

Le chapitre 8 (expérimentations) page 87 permet de retrouver un lot de tests réalisés en laboratoire clos, qui peuvent être facilement reproduits, et qui ont pour objectif de dépasser la théorie en allant vérifier ce qui est exploitable ou non.

Ce contenu s'adresse à la fois aux personnes qui n'ont aucune connaissance en IPv6 et à ceux qui en ont déjà une bonne maîtrise, mais qui n'ont pas une idée claire de ce qui est réellement utilisable actuellement.

Des notions de réseaux sont un prérequis essentiel : bien que certains aspects soient rappelés, ce document n'a pas pour objectif d'aborder des notions qui existent déjà en IPv4 et qui sont simplement retranscrites en IPv6. Il s'intéresse particulièrement aux expériences utilisateurs, et à la migration des parcs finaux, sans s'attarder précisément sur des notions comme le routage dynamique. Le réseau Lothaire pour lequel ce document a été conçu propose d'ores et déjà un routage et des plages d'adresses IPv6 à ses correspondants, mais constate un manque de motivation pour exploiter cette possibilité.

Pour ceux qui ne souhaitent pas perdre de temps et qui veulent directement savoir s'il est possible dès 2012 de **passer tout un réseau en IPv6 en désactivant totalement la couche IPv4**, se reporter à la section dédiée au NAT64/DNS64 (section 5.4.3 page 68) et à la conclusion (page 123).

1. http://en.wikiquote.org/wiki/Tenth_Doctor

1

Pourquoi l'IPv6

« *640K ought to be enough for anybody.* » [1] - Bill Gates (1981)

1.1 Épuisement des adresses IPv4

Jusqu'à 1985, les numéros de téléphones français comportaient sept chiffres. Cette numérotation permettait théoriquement à 10 millions de personnes de communiquer ensemble, pour plus de 65 millions [2] de français recensés actuellement. C'est parce que les opérateurs n'ont pas vu arriver le succès de la téléphonie personnelle qu'ils ont dû passer à huit chiffres en 1985 puis à dix en 1996, pour pouvoir accueillir plus de monde.

De la même façon qu'un numéro de téléphone doit être propre à chaque abonné, une adresse IP ne peut pas être dupliquée sur le réseau Internet. Dès 1981, l'IPv4 (RFC 791) sur 32 bits que nous utilisons encore aujourd'hui apparaît, permettant théoriquement à plus de 4 milliards de machines de communiquer entre elles, pour 7 milliards [3] d'humains.

Le réseau Internet se limitant à l'époque aux grandes universités et aux armées, le nombre de possibilités parait astronomique. Ainsi la même année, l'IANA (*Internet Assigned Numbers Authority*) qui est en charge de la distribution de ces adresses dans le monde entier, décide de les répartir par classes (RFC 791) en utilisant leurs quatre premiers bits comme discriminant. À titre d'exemple, attribuer une classe A à une seule université ou entreprise revenait à lui mettre à disposition plus de 16 millions d'adresses, qui devenaient de ce fait indisponibles pour le reste du monde, qu'elles soient utilisées ou non.

C'est à cause de ce découpage grossier que le réseau Internet connut dès 1993 une première mesure pour enrailler l'épuisement des adresses disponibles. La notation CIDR (*Classless Inter-Domain Routing*)

1. http://www.wired.com/politics/law/news/1997/01/1484
2. Recensement du mois de janvier 2012.
3. Chiffre du 31 octobre 2011, selon les Nations Unies.

et les notions de préfixes réseaux avec le slash, que nous manipulons toujours aujourd'hui, sont les solutions qui ont été proposées par l'IETF (*Internet Engineering Task Force*) pour résoudre ce problème (RFC 1338). En ayant la possibilité de proposer des blocs d'adresses à taille variable, l'IANA peut dès lors commencer à économiser les adresses IPv4 en n'attribuant que le nombre nécessaire d'adresses aux demandeurs.

L'année 1993 est marquée par un autre événement : l'apparition du premier navigateur web graphique. Les portes d'Internet s'ouvrent alors au grand public, qui l'investira avec plus d'engouement que quiconque n'aurait pu le prévoir. Il faut dès lors trouver une nouvelle solution pour prévoir un épuisement qui recommence à inquiéter.

Ainsi, en février 1996, la RFC 1918 instaure le concept de NAT (*Network Address Translation*) et d'adresses privées. Ces dernières ne sont plus uniques et ne sont, par conséquent, plus routables sur le réseau Internet. Derrière une seule IP publique, des milliers de machines en adressage privé : l'économie d'adresses est faramineuse. Mais dès les années 90, personne n'est dupe, c'est une renumérotation à l'échelle mondiale qui sera nécessaire pour accueillir tous les besoins futurs, à l'instar de ce qui s'est fait dans la téléphonie.

À la fin de l'année 1998, l'IETF propose les premières spécifications finalisées de l'IPv6 (la version 5 étant expérimentale), en proposant des adresses quatre fois plus longues. Non seulement ce système de numérotation permet d'adresser plus de machines par mètre carré qu'il n'est physiquement possible d'en placer, mais il apporte un lot étonnant de nouveautés en terme de configuration et de sécurité, que nous aborderons dans ce document.

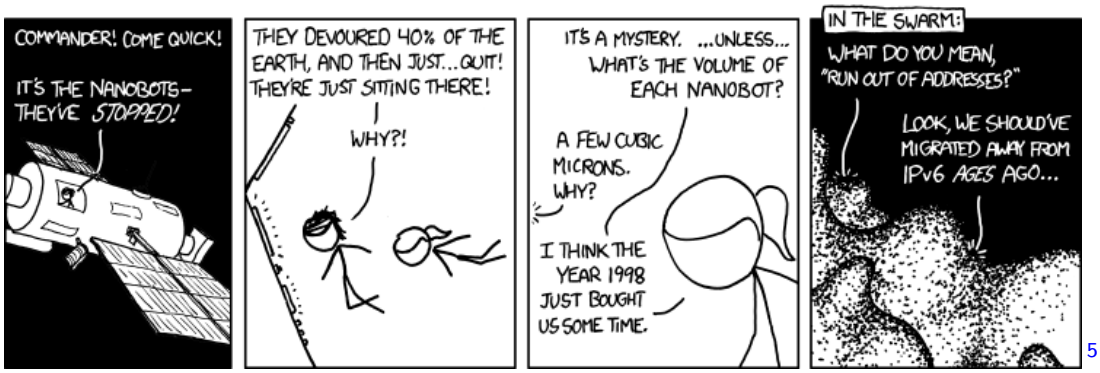
Si les renumérotations dans la téléphonie française n'ont jamais posé de problème, c'est principalement parce que le nombre d'acteurs commerciaux concernés par ce changement est particulièrement réduit. Changer de numérotation IP impose de mettre d'accord des centaines de milliers d'entreprises, qui devront toutes faire les efforts nécessaires pour adapter les logiciels et les équipements, engageant ainsi des flux financiers importants qui n'auront pourtant aucun impact direct sur leurs ventes. S'engager pour le futur d'un intérêt commun, à l'heure où la bourse se négocie à la seconde pour des intérêts privés, relève encore du courage de quelques grosses entreprises. Stéphane Bortzmeyer propose une autre approche de la situation : « *Déployer IPv6 coûte à celui qui le déploie, ne pas le déployer coûte équitablement à tout le monde. Dans un régime capitaliste, le choix est vite fait.* » [4].

Le blogger réputé pousse l'analyse plus loin : « *Pourtant, ce n'est pas une simple question d'argent. La non-migration vers IPv6 coûte très cher, notamment en temps passé à faire fonctionner les applications malgré le NAT, en complexité due à l'existence de deux domaines d'adressage, le privé et le public, en temps passé à remplir des papiers pour la bureaucratie des RIR, qui limite ainsi la consommation d'adresses IPv4, en lignes de code dans les applications SIP ou pair-à-pair pour arriver à contourner l'absence d'adresses globalement uniques. Le coût global de ces mesures est sans doute bien supérieur à celui d'une migration vers IPv6.* ».

Le RIR (*Regional Internet Registry*) de l'Asie-Pacifique a déjà attribué son dernier bloc IPv4, et la pénurie mondiale est de plus en plus pesante avec l'émergence de nouveaux supports comme les ordiphones. La solution d'un NAT à l'échelle d'un opérateur comme elle est pratiquée dans ce secteur est d'autant plus inquiétante qu'elle nuit gravement à la neutralité du Net, en interdisant à quiconque d'être accessible depuis l'extérieur.

4. http://www.bortzmeyer.org/ipv6-et-l-echec-du-marche.html

De la même façon que nous n'avons pas eu le choix de changer de numéro de téléphone, l'adoption massive de l'IPv6 est inévitable et doit concerner chaque acteur de l'informatique à quel niveau qu'il soit, pour réussir tous ensemble à construire un Internet plus sain et plus performant.



[5]

Ajoutons enfin que certains s'amusent à rappeler qu'il reste encore des millions d'IPv4 inutilisées dans le monde, et qu'il ne s'agit que d'un prétexte pour faire monter leurs prix et faire renouveler du matériel. Il y a effectivement beaucoup d'institutions qui conservent encore des plages énormes d'adresses IPv4, mais si elles ne sont pas utilisées, elles ne sont pas pour autant redistribuables. D'une façon générale, la pénurie accroit toujours les inégalités, ce qui permet aux USA de ne pas encore voir la fin de l'IPv4 alors que l'Asie en est proche ou que de petits FAI n'arrivent plus à se faire attribuer de blocs PI [6] (*Provider Independent*).

La section suivante rappelle que le passage à l'IPv6 ne concerne pas seulement la pénurie d'IP, mais promet aussi à terme un réseau plus rapide, plus performant et moins coûteux.

1.2 Accès directs sans réécriture

Il existe un certain nombre de problèmes engendrés par les réécritures des NAT :

Difficultés pour la voix sur IP : les accès aux téléphones IP se font au prix d'acrobaties avec le NAT, qui empêche d'une façon générale le vrai pair-à-pair.

Difficultés pour la visioconférence : aux problèmes similaires sus-évoqués, nous pouvons rajouter l'engorgement des routeurs qui doivent traduire des milliards de paquets, ainsi que les problèmes récurrents de négociation de ports bilatéraux.

Difficultés d'accès aux serveurs internes : bien que cet aspect ait tendance à être utilisé pour renforcer la sécurité, il est très souvent contraignant (citons l'exemple de multiples serveurs HTTP derrière une seule adresse IP publique, qui nécessitent un reverse-proxy coûteux en performances ou la mise en place de VPN à administrer).

Chiffrement quasiment impossible : si une communication devait être chiffrée de bout en bout avec un NAT, il faudrait alors que l'équipement intermédiaire soit capable de le déchiffrer pour récupérer les informations des entêtes.

5. http://xkcd.com/865

6. Voir à ce sujet les conditions du RIPE concernant leur dernier /8 qui sera découpé en PA (*Provider Aggregable*) et qui condamnera donc très bientôt les nouveaux petits FAI à être dépendants des LIR (*Local Internet Registry*) : http://www.ripe.net/ripe/docs/ripe-553#-----use-of-last-8-for-pa-allocations

Gestion des journaux systèmes : utiliser une adresse publique unique pour un parc entier de machines nécessite d'être capable en permanence d'associer une date et une heure à une machine de l'adressage privé, afin d'être à même de répondre aux requêtes judiciaires.

1.3 Conflits et collisions

Utiliser des adresses privées signifie utiliser une numérotation qui n'est pas unique. Cette pratique pose un certain nombre de problèmes :

Mise en place d'un VPN entre sites : si par malheur les deux sociétés qui souhaitent s'offrir des accès VPN utilisent le même préfixe de réseau privé pour adresser leurs machines, l'une des deux devra renuméroter son parc ou utiliser une technique de traduction d'adresses bidirectionnelle extrêmement lourde.

Fusion de deux sites : le même problème que précédemment se pose, et la seule solution pour ne pas renuméroter toutes ses machines (ce qui est souvent de l'ordre de l'inenvisageable) consiste à utiliser un double NAT, qui deviendra rapidement une plaie au quotidien.

Broadcasts intempestifs : les paquets diffusés à l'ensemble du réseau sont devenus monnaie courante en IPv4, ce qui entraîne des surcharges du réseau, des collisions plus fréquentes, et des boucles sur les réseaux complexes.

1.4 Les solutions et nouveautés

Toutes ces problématiques n'ont pas lieu d'être avec le modèle initial du réseau Internet, qui consiste à adresser toutes les machines au même rang, en leur donnant toutes la possibilité d'être un nœud supplémentaire sur le réseau mondial. Alors que la pénurie des adresses IPv4 nous a forcé à adapter notre façon de concevoir Internet, à un tel point que les nouvelles générations d'informaticiens ne connaissent plus que celle-ci, l'IPv6 nous permettra de revenir à ces fondamentaux d'hier qui sont une solution efficace aux problèmes d'aujourd'hui et de demain.

Notons dès à présent que **le NAT est à différencier de la notion de pare-feu** : s'abstraire de la notion d'adresses privées ne signifie en aucun cas permettre systématiquement des accès non-contrôlés aux machines d'un réseau depuis l'extérieur.

Outre l'abolition des NAT et le nombre quasi-illimité d'adresses, l'IPv6 offre d'autres solutions qui en découlent parfois :

Plusieurs adresses par interface : c'était déjà possible en IPv4, mais ça se généralise avec l'IPv6, permettant ainsi de faciliter les migrations, d'accroître la sécurité et de faciliter l'hébergement de multiples services.

Adresses de lien local uniques : grâce à des mécanismes qui seront détaillés plus tard dans ce document, toutes les adresses d'un réseau qui n'a pas pour vocation de se relier à Internet se génèrent entièrement d'elles-mêmes, de façon assurément uniques (équivalent du zéroconf généralisé).

Suppression du broadcast : il est en réalité remplacé par une adresse multicast à laquelle on est libre d'être inscrit ou non, et par une collection d'autres adresses multicasts utilisées en priorité pour

certains usages courants.

Présence de IPsec systématique : la fonctionnalité n'est pas la nouveauté, mais plutôt l'assurance qu'elle sera supportée par tout matériel ou logiciel supportant l'IPv6.

Entêtes IP simplifiés : non seulement le champ *checksum* a disparu (il délègue le travail de vérification aux couches supérieures) et n'a plus besoin d'être recalculé systématiquement par les routeurs, mais les entêtes ont été réduites, allégeant ainsi d'autant la charge des réseaux. Une autre conséquence de cette délégation du calcul du *checksum* est qu'il sera calculé au niveau de TCP pour l'ensemble du paquet plutôt que pour l'entête seul. Il devient également obligatoire pour UDP.

Mobilité et renumérotation : les mécanismes mis en place par l'IPv6 permettent facilement de distinguer les préfixes réseau des identifiants d'interface.

Multicast valorisé : des groupes multicast génériques permettent de contacter des machines par type, en envoyant par exemple une requête de découverte du DHCP à tous les routeurs en multicast, plutôt qu'à l'ensemble des machines du réseau en broadcast.

Autoconfiguration *stateless* : l'IPv6 permet à une machine de s'insérer dans un réseau sans aucune configuration manuelle et sans DHCP à gérer.

1.5 Des paquets plus cohérents

Alors que la structure des paquets IPv4 a été conçue à une époque où personne n'imaginait l'ampleur qu'allait prendre le modèle TCP/IP, la norme IPv6 est une occasion de corriger les erreurs qui ont été commises. La figure 1.1 donne un aperçu du nouveau format des paquets.

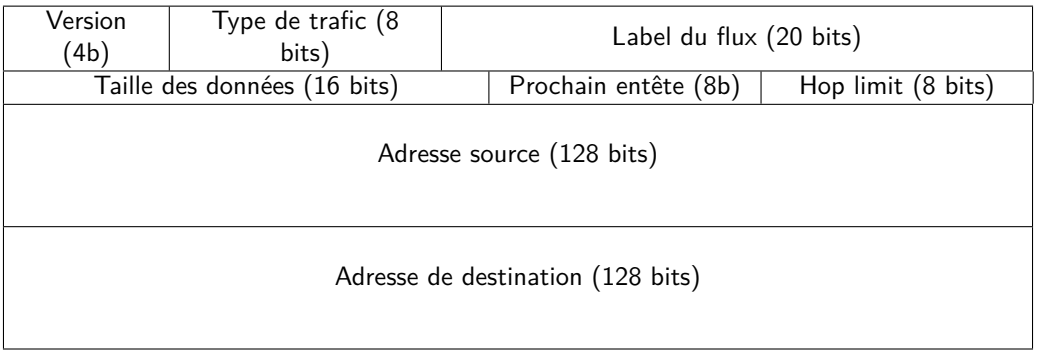


Figure 1.1 – *Paquet IPv6.*

Les différences importantes sont :

- Disparition du champ *IP Header Length* (la taille des entêtes est fixe).
- Le TTL (*Time To Live*) est renommé en *Hop Limit* puisque c'était déjà son usage en IPv4.[7]
- Aucun champ *checksum*, qui nécessitait un nouveau calcul à chaque routage en IPv4 à cause du TTL qui était inclus dedans (la tâche est déléguée à la couche transport).
- La taille des données n'inclut pas la taille des entêtes.

7. La RFC 791 indiquait initialement pour le TTL : « *This field indicates the maximum the datagram is allowed to remain in the internet system. [...] The time measured in units of seconds.* ».

– Des entêtes optionnels peuvent être ajoutés à la suite de l'entête, notamment pour le numéro de fragmentation.

Un algorithme de découverte du MTU de bout en bout (*Path MTU Discovery*, RFC 1191) est destiné à être déployé massivement, pour éviter toute fragmentation, celle-ci réduisant considérablement le débit (si un seul paquet est perdu, tout doit être retransmis). En cas de fragmentation, celle-ci n'est plus du ressort des routeurs intermédiaires, qui doivent se contenter de retourner un message ICMPv6 *Packet Too Big*. L'émetteur aura alors pour charge de reproposer le paquet qu'il aura lui-même découpé, permettant ainsi aux routeurs intermédiaires de nécessiter moins de puissance.

La MTU minimale est de 1280 octets (contre 68 en IPv4) et la taille maximale d'un paquet est de 65535 octets, comme en IPv4 (avec l'option *jumbogram* de la RFC 2675, qui permet de l'augmenter jusqu'à 4 Go).

1.6 État du déploiement de l'IPv6 en France

1.6.1 FAI publics

L'opérateur Nerim semble être le premier à avoir proposé de l'IPv6, en proposant des /48 natifs systématiques à ses abonnés dès 2003 [8]. Pour résoudre les cas de figure où l'accès natif est impossible, il propose aussi des tunnels à encapsulation sur de l'IPv4.

Vient ensuite l'opérateur Free qui propose des adresses IPv6 très attendues [9] dès 2007 pour les abonnés dégroupés. Dans le communiqué de presse [10], Iliad précise qu'il est « *l'un des premiers opérateurs dans le monde à faire évoluer son réseau* ». Comme nous le verrons dans la partie dédiée au 6rd, cette affirmation n'est pas tout à faire vraie puisqu'ils utilisent un mécanisme d'encapsulation qui leur permettent justement de ne pas faire évoluer leur réseau. Ils se limitent à offrir un /64, qui permet tout de même bien plus d'adresses qu'un particulier ne pourra jamais avoir besoin. Toutefois, en considérant que les 64 derniers bits d'une adresse doivent être réservés pour l'autoconfiguration *stateless* (abordée dans la suite de ce document), ce préfixe ne permet pas de faire des sous-réseaux depuis une ligne Free.

Le 26 mai 2011, c'est une certaine Chloé qui annonce l'ouverture d'une période de bêta-tests du côté de SFR-Neufbox [11]. On apprend dans une news Cisco [12] que SFR aurait utilisé le CGv6 [13] avec un ASR 1000 pour migrer son réseau. On peut donc en déduire que SFR se contente aussi d'encapsuler de l'IPv6 sur de l'IPv4.

Présent depuis 2009 sur le réseau MPLS [14] (groupe de l'IETF chargé de standardiser un certain nombre de techniques de transport sur des trames de niveau 2), Orange ne propose toujours pas d'IPv6 à ses abonnés. Malgré tout il ne l'ignore pas totalement sur ses infrastructures, puisqu'il propose des VPN

8. http://www.nerim.fr/ipv6
9. http://ipv6pourtous.free.fr/rani/
10. http://www.iliad.fr/presse/2007/CP_IPv6_121207.pdf
11. http://atelier.sfr.fr/beta-tests/la-neufbox-sfr-passe-a-l-ipv6
12. http://newsroom.cisco.com/press-release-content?type=webcontent&articleId=358080
13. http://www.cisco.com/en/US/netsol/ns1017/networking_solutions_solution_category.html
14. http://datatracker.ietf.org/wg/mpls/charter/

IPv6 à destination des entreprises (voir la jolie vidéo [15]). Pour la suite, un projet plus large semble être dans les rails dans le cadre de « *Conquêtes 2015* » comme en témoigne son communiqué de presse [16]. L'accès aux particuliers serait prévu pour 2013 avec une nouvelle Livebox en 2012. Bien qu'il soit le plus gros opérateur français, l'« Internet by Orange » lui vaut une fois de plus d'être à la traîne derrière ses concurrents.

1.6.2 Réseaux de recherche

Renater, le réseau français pour l'éducation et la recherche, est aux coude-à-coude avec Nerim pour l'innovation en France, puisqu'il propose aussi dès 2003 de l'IPv6 sur l'ensemble de son réseau [17]. Il précise que tous les routeurs sont en double pile, et qu'il utilise le réseau GÉANT pour communiquer en IPv6 avec le reste du monde. Environ 160 préfixes /48 ont été alloués en sa qualité de LIR (*Local Internet Registry*). Il propose également une carte de France publique de l'état de ses connexions IPv6 en temps réel [18].

Le réseau fédérateur GÉANT propose de l'IPv6 depuis 2002, et dresse un bilan de la situation de l'IPv6 dans plusieurs pays du monde [19].

1.6.3 Sites web populaires

La figure 1.2 propose une analyse du top 100 des sites les plus visités par les français en mai 2012 selon Alexa [20], en colorisant en vert les sites accessibles en IPv6 [21].

Seuls 10% des sites populaires sont accessibles en IPv6, dont 50% provient des serveurs de Google. Certains sites comme *ovh.com* sont accessibles en IPv6 via une adresse spécifique (*ipv6.ovh.com*), ce qui ne rend leur site web accessible que pour les initiés.

Le constat n'est pas très encourageant, mais migrer un site web en IPv6 n'est pas techniquement compliqué. Il tient donc probablement plus de la fainéantise des éditeurs que de l'incompétence technique ou du coût financier de la migration. Si tous les internautes disposaient de l'IPv6 et que tous les datacenters le mettaient aussi à disposition, il y a fort à parier que le tableau changerait très rapidement.

Cette liste à elle-seule justifie l'importance de la nécessité de faire cohabiter l'IPv4 avec l'IPv6.

1.6.4 *IPv6 day*

En 2011 a eu lieu la première journée mondiale de l'IPv6 (logo en figure 1.3), organisée par l'Internet Society aidée par de grosses compagnies. Alors qu'elle était destinée à encourager les acteurs du monde

15. http://ipv6.orange-business.com
16. http://mobile.orange.fr/content/ge/high/v2_a_propos_d_orange/cp/244685.pdf
17. http://www.renater.fr/deploiement-ipv6-sur-le-reseau-renater?lang=fr
18. http://pasillo.renater.fr/weathermap/weathermap_france_ipv6.html
19. http://www.geant.net/Network/NetworkTopology/Pages/IPv6.aspx
20. http://www.alexa.com/topsites/countries/FR
21. `while read site; do (dig AAAA +short +tcp $site | grep -v \\. > /dev/null) && echo "\\item {\\color{green} $site}" || echo "\\item {\\color{red} $site}"; done < sites.txt >> sites_ip6.txt`

1. google.fr
2. facebook.com
3. google.com
4. youtube.com
5. yahoo.fr
6. fr.wikipedia.org
7. live.fr
8. t.co
9. leboncoin.fr
10. www.orange.fr
11. www.free.fr
12. fr.linkedin.com
13. ebay.fr
14. twitter.com
15. commentcamarche.net
16. viadeo.com
17. amazon.fr
18. lemonde.fr
19. dailymotion.com
20. over-blog.com
21. blogger.com
22. figaro.fr
23. msn.fr
24. pagesjaunes.fr
25. wordpress.com
26. lequipe.fr
27. blogger.fr
28. sfr.fr
29. programme-tv.net
30. allocine.fr
31. linternaute.com
32. laredoute.fr
33. www.googleusercontent.com
34. apple.com
35. vente-privee.com
36. microsoft.com
37. pole-emploi.fr
38. ovh.net
39. tumblr.com
40. adcash.com
41. leparisien.fr
42. seloger.com
43. tf1.fr
44. xhamster.com
45. voyages-sncf.com
46. ovh.com
47. paypal.com
48. 20minutes.fr
49. jeuxvideo.com
50. bing.fr
51. priceminister.com
52. skyrock.fm
53. www.doctissimo.fr
54. deezer.com
55. cdiscount.fr
56. pinterest.com
57. journaldunet.com
58. boursorama.fr
59. badoo.com
60. societe.com
61. lexpress.fr
62. aufeminin.com
63. nouvelobs.com
64. flickr.fr
65. youporn.com
66. becoquin.com
67. meteofrance.com
68. fnac.com
69. credit-agricole.com
70. liberation.fr
71. bnpparibas.com
72. clubic.com
73. babylon.com
74. advertstream.com
75. laposte.net
76. caisse-epargne.fr
77. 01net.com
78. canalplus.fr
79. ad6media.fr
80. voila.fr
81. canalblog.com
82. webrankinfo.com
83. new.livejasmin.com
84. rueducommerce.fr
85. pornhub.com
86. xvideos.com
87. lepoint.fr
88. societegenerale.fr
89. vivastreet.co.uk
90. reverso.net
91. wordreference.com
92. adserverpub.com
93. labanquepostale.fr
94. conduit.com
95. mobile.free.fr
96. adobe.com
97. banquepopulaire.fr
98. pixmania.com
99. groupon.com
100. amazon.com

Figure 1.2 – *Les 100 sites les plus visités par les français et l'IPv6 (sites compatibles en vert).*

entier à tester leurs infrastructures en IPv6, la version 2012 avait pour objectif de les encourager à proposer leurs services en IPv6 de façon durable.

Figure 1.3 – *Logo de la journée mondiale pour l'IPv6 le 06/06.*

En France, des FAI comme Orange ou SFR ont participé à cette journée mondiale aux côtés d'acteurs mondiaux comme Cisco [22] ou Google [23], qui aura maintenant lieu tous les ans à la date symbolique du 06/06.

La figure 1.4 page 11 semble prouver qu'au niveau mondial, que ce soit grâce à cette initiative

22. http://www.cisco.com/web/solutions/trends/ipv6/world_day.html
23. http://www.google.com/intl/en/ipv6

populaire ou non, l'utilisation d'IPv6 dans le monde explose depuis ces dernières années (mesures réalisées par Google [24]).

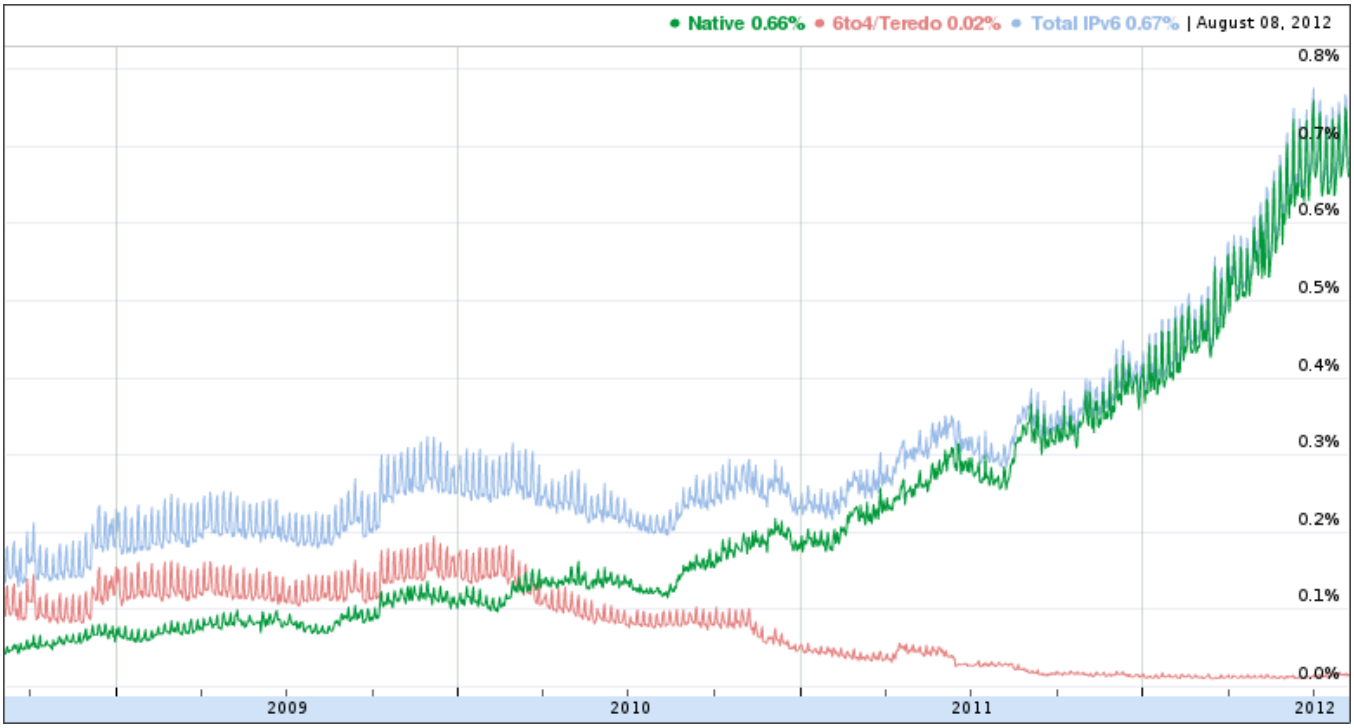


Figure 1.4 – *Utilisation de l'IPv6 au travers du temps, mesurée par Google.*

24. http://www.google.com/ipv6/statistics.html

2

Adressage

« *I don't know if it's what you want, but it's what you get.* » [1] - Larry Wall (1990)

2.1 Conventions d'écriture

Les nouvelles adresses IP comporteront 128 bits, soit 16 octets. Pour faciliter leur lecture, celles-ci sont découpées en blocs de 16 bits séparés par le symbole deux-points :

```
2001:0db8:0000:0000:0000:abcd:def0:1234
```

Tous les zéros en tête de bloc peuvent être supprimés, et les quadruplets de zéros réduits à un seul *nibble* :

```
2001:db8:0:0:0:abcd:def0:1234
```

Plus simple encore, une suite (de 1 à 7) de blocs de zéros peut être induite par un double deux-points :

```
2001:db8::abcd:def0:1234
```

Cette notation indique aux interpréteurs de remplacer ce double symbole par autant de zéros qu'il n'en faut pour arriver aux 128 bits nécessaires, sur la totalité de l'adresse donnée. Par souci d'unicité des possibilités d'extension, le double deux-points ne peut être utilisé qu'à un seul endroit dans l'adresse.

La notion historique de classes a totalement disparue, au profit de l'utilisation exclusive des préfixes et de la notation CIDR avec le slash et le masque, déjà utilisées en IPv4. Les masques canoniques disparaissent donc, au grand bonheur des administrateurs.

Ainsi, pour un préfixe sur 60 bits tel que `2001:0db8:0000:ba3` :

1. http://groups.google.com/groups?selm=10502@jpl-devvax.JPL.NASA.GOV&hl=en

```
2001:0db8::ba30:0:0:0:0/60
ou
2001:db8:0:ba30::/60
ou encore
2001:0db8:0000:ba30:0000:0000:0000:0000/60
```

Comme en IPv4, une adresse d'interface peut être suivie de son masque de réseau grâce à la notation CIDR.

Dans un contexte de requête HTTP, l'adresse IPv6 doit obligatoirement être entre crochets pour différencier le port du reste de l'adresse (ainsi que dans d'autres cas d'ambiguïté, comme avec SCP) :

```
HTTP : [2001:db8::abcd:def0:1234]:8080
SCP  : [2001:db8::abcd:def0:1234]:/etc/passwd
```

Concernant le calcul des sous-réseaux IPv6, l'outil *sipcalc* apporte une précieuse aide :

```
$ sipcalc -6 2001:db8::/48 -S /56
-[ipv6 : 2001:db8::/48] - 0

[Split network]
Network               - 2001:0db8:0000:0000:0000:0000:0000:0000 -
                        2001:0db8:0000:00ff:ffff:ffff:ffff:ffff
Network               - 2001:0db8:0000:0100:0000:0000:0000:0000 -
                        2001:0db8:0000:01ff:ffff:ffff:ffff:ffff
Network               - 2001:0db8:0000:0200:0000:0000:0000:0000 -
                        2001:0db8:0000:02ff:ffff:ffff:ffff:ffff
Network               - 2001:0db8:0000:0300:0000:0000:0000:0000 -
...
```

La section 2.9 page 24, concernant les bonnes pratiques, apporte des informations complémentaires sur l'utilisation de ces adresses.

2.2 Adresses unicast

2.2.1 Généralités

Il s'agit du même type d'adresse (RFC 4291, section 2.5) que son homologue IPv4. C'est donc le type d'adresse le plus classique, composé d'un préfixe réseau à longueur variable suivi d'un identifiant d'interface (parfois dérivé des adresses MAC, sur 64 bits).

La figure 2.1 illustre le mode de diffusion.

On distingue plusieurs sous-catégories.

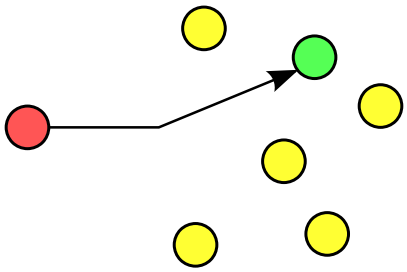


Figure 2.1 – *Représentation schématique d'une diffusion unicast.*

2.2.2 Link-local Unicast

Adresses de lien local, non-routables en local comme sur Internet. C'est un peu l'équivalent des adresses `169.254/16` du zéroconf de l'IPv4.

Elles utilisent toutes le préfixe `fe80::/10`, elles sont donc souvent directement reconnaissables par leur premier bloc. Elles sont systématiquement générées lors de l'utilisation de l'autoconfiguration *stateless*.

2.2.3 Unique Local Unicast

Ces adresses peuvent être utilisées sur un lien local comme les précédentes, mais qui pourra être partagé entre plusieurs sites par un tunnel. Elles ne sont pas non plus routables sur Internet, et c'est la catégorie qui se rapproche le plus des adresses privées IPv4 (RFC 1918).

Leur différence par rapport au type étudié ci-avant dépend de l'utilisation d'un algorithme censé assurer leur unicité en cas de branchement avec un autre site, détaillé dans la RFC 4193.

Elles utilisent toutes le préfixe `fc00::/7`, mais avec le huitième bit en partant de la gauche positionné à `1` si le préfixe est défini localement. Puisque la valeur `0` n'est pas possible actuellement (usage futur), elles sont reconnaissables par leur premier bloc qui commence systématiquement par `fd`.

Des outils comme UltraTools[2] permettent de les générer sans se soucier du fonctionnement de l'algorithme détaillé dans la RFC.

Ce type d'adresse remplace le type site-local unicast, rendu obsolète.

2.2.4 Global Unicast

Tout ce qui ne correspond ni à l'une des autres catégories d'adresses unicast, ni à une adresse multicast ou anycast, est une adresse globale. Il s'agit des adresses qui sont uniques dans le monde, et qui sont par conséquent routables sur Internet.

Elles se composent d'un préfixe de routage global, suivi du préfixe de sous-réseau et de l'identifiant

2. http://ultratools.com > UltraTools > IPv6 Tools > Local IPv6 Range Generator

d'interface. L'IANA fournit la liste des préfixes[3] affectés aux différents RIR, qui permettent donc de classer les IP rencontrées sur le réseau mondial par région du monde.

Si le préfixe `2001:db8::/32` semble appartenir à l'APNIC (Asie-Pacifique), il n'est en réalité pas routable sur Internet puisqu'il s'agit des adresses à utiliser pour les exemples des documentations (RFC 3849).

2.3 Adresses multicast

2.3.1 Généralités

Ce type d'adresse (RFC 4291, section 2.7) correspond précisément à son homologue IPv4. Il s'agit d'adresses virtuelles gérées par les routeurs, qui redistribuent des paquets à tous les membres inscrits dans un groupe, à l'instar du fonctionnement d'une liste de diffusion de courriels.

La figure 2.2 illustre le mode de diffusion.

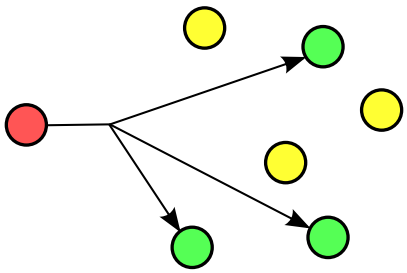


Figure 2.2 – *Représentation schématique d'une diffusion multicast.*

Elles utilisent toutes le préfixe `ff00::/8`[4]. Elles sont donc reconnaissables par leur premier bloc qui commence systématiquement par `ff`.

2.3.2 Structure

Le format d'une IP multicast est donné en figure 2.3.

8 bits	4 bits	4 bits	112 bits
`1111 1111`	Drapeaux : `0RPT`	Portée	Identifiant de groupe

Figure 2.3 – *Format d'une IP multicast.*

Les quatre bits qui suivent le préfixe désignent les drapeaux (*flags*) :

`0` **(*zéro !*) :** Il s'agit d'un zéro immuable.

3. http://iana.org/assignments/ipv6-address-space/ipv6-address-space.xml
4. Liste complète : http://en.wikipedia.org/wiki/Multicast_address#IPv6

`T` **(*Temporaire ?*) :** Il doit être à 0 lorsque c'est une adresse affectée de façon permanente (*well-known address*) par l'IANA, sinon il est à 1.

`P` **(*issue du Préfixe ?*) :** Il doit être à 0 si l'adresse multicast ne correspond pas au préfixe diffusé sur le réseau, ce qui est obligatoirement le cas si le drapeau `T` est lui-même à 0, sinon il sera à 1 (RFC 3306).

`R` **(*utilise un Rendez-vous ?*) :** Il sera à 1 si l'adresse multicast utilise un point de « rendez-vous » [5] (auquel cas `P` est à obligatoirement à 1, ce qui signifie que `T` aussi), et sera donc la plupart du temps à 0.

Les quatre bits qui suivent les drapeaux correspondent à la portée de l'adresse (*scope*), qui peuvent limiter son rayonnement à l'interface (1) pour les tests, au lien local (2), au site correspondant au réseau local (5), à Internet (e) ou à des zones plus variées décrites dans la RFC 4291 [6].

Cette façon de faire permet de contacter des services efficacement en ajustant le type d'accès. Ainsi, en définissant l'identifiant 123 pour tous les serveurs NTP, on aura la possibilité de contacter différents niveaux de serveurs selon le préfixe associé (tableau 2.1).

Adresse	Serveurs NTP contactés
`ff01::123`	Ceux qui sont sur la même interface
`ff02::123`	Ceux du même lien
`ff05::123`	Ceux du LAN
`ff0e::123`	Tous y compris sur Internet

Table 2.1 – *Exemple d'utilisation de la portée des adresses multicast avec NTP.*

2.3.3 Adresses utiles

Un certain nombre d'adresses multicast sont normalisées par l'IETF (drapeau `T` positionné à 0), et sont documentées dans la RFC 2461 [7].

Les plus utilisées (portée de lien local) sont référencées dans le tableau 2.2.

Les systèmes Debian (par exemple) associent par défaut un domaine à ces adresses (*/etc/hosts*) :

```
1 # The following lines are desirable for IPv6 capable hosts
2 ::1     ip6-localhost ip6-loopback
3 fe00::0 ip6-localnet
4 ff00::0 ip6-mcastprefix
5 ff02::1 ip6-allnodes
6 ff02::2 ip6-allrouters
```

5. http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6552/whitepaper_c11-508498.html
6. Ou sur http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml.
7. Ou sur http://en.wikipedia.org/wiki/Multicast_address#IPv6.

Nom	Adresse	Équivalent IPv4	Fonction
all-nodes	`ff02::1`	`224.0.0.1`	Tous les nœuds et routeurs du lien local (utilisée par exemple pour les interfaces qui n'ont pas encore d'adresse mais qui veulent recevoir une réponse, notamment pour savoir s'il y a duplication dans l'algorithme DAD, abordé par la suite)
all-routers	`ff02::2`	`224.0.0.2`	Tous les routeurs du lien local (utilisée par exemple pour solliciter une annonce de préfixe sur le réseau)
solicited-node	`ff02::1:ff*`	Aucun	Permet de contacter un nombre restreint de machines, notamment pour faire une association MAC-IP (protocole NDP) plutôt que de broadcaster comme le faisait l'ARP

Table 2.2 – *Adresses multicast couramment utilisées (portée locale).*

2.3.4 Trame ethernet

L'adresse ethernet (MAC) d'une trame à destination d'un groupe multicast est dérivée de l'adresse IP du groupe (RFC 5342 et 2464), comme décrit dans la figure 2.4.

16 bits	32 bits
`ff02::1:ff00:0`	32 derniers bits de l'adresse IPv6 de l'interface

Figure 2.4 – *Format d'une adresse ethernet multicast.*

La figure 2.5 illustre cette conversion.

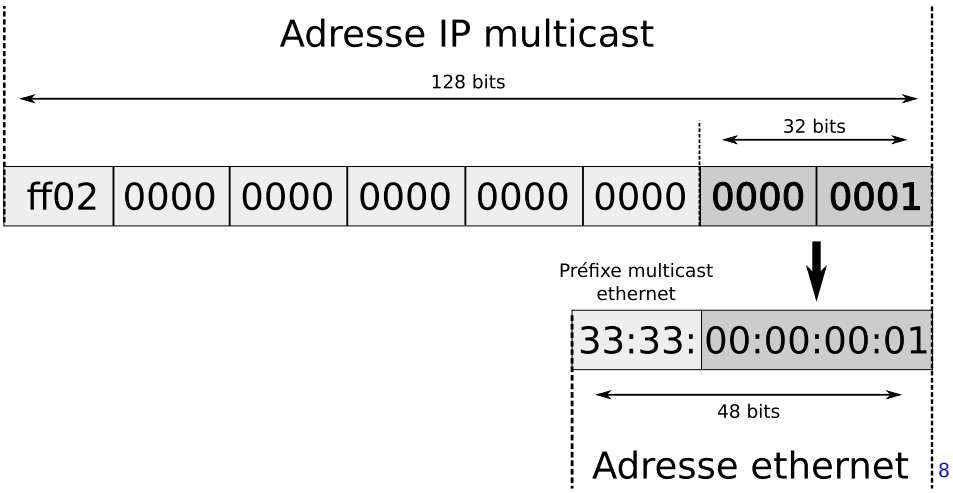


Figure 2.5 – *Conversion d'une adresse IP multicast en adresse ethernet multicast (adresse* all-nodes*).*

2.3.5 Abonnements systématiques

Étant données les notions étudiées ci-avant, les groupes multicasts utilisés dans les différents algorithmes nécessitent que les machines rejoignent des groupes automatiquement, parfois en fonction de la nature de leur rôle sur le réseau.

Exemple de groupes multicast IPv6 automatiquement rejoints par une Debian en autoconfiguration *stateless* (par défaut) :

```
# Adresse MAC qui a servie pour l'autoconfiguration des adresses IPv6
$ ip link show dev eth0
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
    UP mode DEFAULT qlen 1000
    link/ether 00:21:70:ec:b4:9b brd ff:ff:ff:ff:ff:ff

# Adresses IPv6
$ ip -6 addr show dev eth0
2: eth0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 2001:660:4503:105:221:70ff:feec:b49b/64 scope global dynamic
       valid_lft 2591946sec preferred_lft 604746sec
    inet6 fe80::221:70ff:feec:b49b/64 scope link
       valid_lft forever preferred_lft forever

# Abonnements multicast
$ ip -6 maddress show dev eth0
2:      eth0
        inet6 ff02::fb
        inet6 ff02::202
        inet6 ff02::1:ffec:b49b users 2
        inet6 ff02::1
```

La signification des inscriptions pour un poste utilisateur est renseignée dans le tableau 2.3 page 20.

Exemple de groupes multicast IPv6 automatiquement rejoints par un routeur Cisco en autoconfiguration *stateless* (par défaut) :

```
# Adresses IPv6 et groupes multicast rejoints
Router# show ipv6 interface fa 0/1

  IPv6 is enabled, link-local address is FE80::217:59FF:FE02:8DB9
  No Virtual link-local address(es):
  Global unicast address(es):
    2001:660:4503:105::2, subnet is 2001:660:4503:105::/64
  Joined group address(es):
    FF02::1
    FF02::2
    FF02::1:FF00:2
    FF02::1:FF02:8DB9
```

La signification des inscriptions pour un routeur est renseignée dans le tableau 2.4 page 20.

Groupe multicast	Équivalent IPv4	Signification
`ff02::fb`	`224.0.0.251`	Écoute des trames mDNS (résolutions DNS sans configuration de serveur DNS)
`ff02::202`	Via l'adresse de broadcast	Écoute des trames de broadcast RPC (*Remote Procedure Calls*)
`ff02::1:ffec:b49b`	Aucun	Adresse *solicited-node* correspondant à la fois à son adresse IP de lien local en `fe80:` et son adresse globale en `2001:`, puisque les 24 derniers bits de chacune sont identiques (il s'agit aussi des 24 derniers bits de l'adresse MAC) - le *users 2* signifie que deux processus écoutent sur la socket correspondante
`ff02::1`	`224.0.0.1`	Adresse *all-nodes*, qui remplace plus ou moins le broadcast (en beaucoup moins sollicitée, et de laquelle on pourrait éventuellement se désinscrire bien que ça ne soit pas du tout conseillé)

Table 2.3 – *Groupes multicast automatiquement rejoints par une Debian.*

Groupe multicast	Équivalent IPv4	Signification
`ff02::1`	`224.0.0.1`	Les routeurs écoutent aussi l'adresse *all-nodes*
`ff02::2`	`224.0.0.2`	Adresse *all-routers*, jointe automatiquement dès lors que le *unicast-routing* global est activé [a] pour l'IPv6
`ff02::1:ff00:2`	Aucun	Adresse *solicited-node* correspondant à l'adresse unicast globale en `2001:`
`ff02::1:ff00:8db9`	Aucun	Adresse *solicited-node* correspondant à l'adresse de lien local en `fe80:`, différente de celle utilisée pour l'adresse globale puisque cette dernière n'a pas utilisé l'autoconfiguration mais a été fixée manuellement

a. https://supportforums.cisco.com/thread/2155384?tstart=0

Table 2.4 – *Groupes multicast automatiquement rejoints par un routeur Cisco.*

Toutes ces adresses multicast sont de type lien local (`ff02:`) : contrairement aux adresses de portée supérieure, elles ne donnent pas lieu à une inscription sur les routeurs, qui ne connaissent donc pas la composition des groupes. Lorsqu'une trame est à destination d'une adresse multicast de ce type (donc adressée à une MAC en `33:33:`), elle est transmise par les switchs sur l'ensemble des interfaces, de la même façon que du broadcast IPv4. Si cette façon de faire ne réduit pas les charges réseau et n'évite pas les problèmes de boucle de niveau 2, elle réduit tout de même la charge des cartes réseaux, puisqu'elles peuvent rejeter le paquet simplement en comparant l'adresse MAC de destination avec leurs tables multicast locales, sans nécessiter de désencapsulation.

2.4 Adresses de broadcast

Le broadcast a été supprimé et ne doit plus être pris en considération dans les plans d'adressage, au profit de l'utilisation massive du multicast. Le groupe multicast qui se rapproche le plus du fonctionnement du broadcast traditionnel correspond au groupe *all-nodes* donné en exemple et censé correspondre à tous les nœuds (mais dont l'inscription est à la discrétion de la machine).

La figure 2.6 illustre le mode de diffusion.

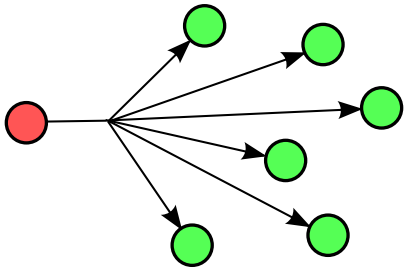


Figure 2.6 – *Représentation schématique d'une diffusion broadcast.*

Avec cette nouvelle approche, l'IPv6 tente de résoudre les problèmes de tempêtes de broadcast, fréquemment rencontrés en version 4. Les groupes prédéfinis permettent de contacter les machines par type de service, en évitant ainsi de charger des machines qui n'ont aucune chance d'être concernées par le message.

2.5 Adresses anycast / réseaux

Les adresses anycast (RFC 4291, section 2.6) ne sont pas différenciables des adresses unicast (seule les interfaces qui les utilisent savent qu'elles sont anycast).

Elles permettent de contacter une machine parmi un lot sans en avoir conscience. Ainsi, ce sera la première qui répondra (souvent la plus proche topologiquement parlant) qui sera utilisée, et les autres ne recevront aucun des paquets transmis. Ce type d'adresse peut donc être affecté à plusieurs interfaces sur un réseau.

La figure 2.7 illustre le mode de diffusion.

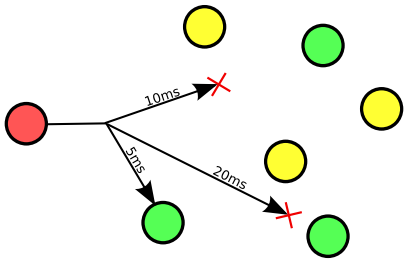


Figure 2.7 – *Représentation schématique d'une diffusion anycast.*

L'une des principales utilisations de cette opportunité concerne les routeurs, en donnant la possibilité de contacter automatiquement le routeur le plus proche d'un sous-réseau donné. Il a été standardisé qu'il suffit de contacter en unicast l'adresse correspondant au préfixe de réseau suivi de zéros (ce qu'on appelait autrefois l'adresse du réseau) pour contacter le routeur qui sera en capacité de nous répondre le plus rapidement possible. C'est une solution pour la redondance des routeurs qui n'est pas obligatoire, mais qui est plus simple à mettre à en œuvre que l'utilisation du HSRP. Sur un routeur Cisco, l'IOS ne respecte pas cette norme par défaut, et se contente uniquement de prévenir du caractère unicast de l'adresse si l'utilisateur tente de l'attribuer à une interface.

2.6 Adresses de boucles locales

Les administrateurs qui ne voient pas beaucoup le soleil et qui ont `127.0.0.1` écrit sur leur paillasson, devront le troquer pour une version plus récente avec `::1` inscrit dessus.

L'adresse `::` correspond logiquement à l'adresse IPv4 `0.0.0.0` et ne sera donc utilisée que pour définir les passerelles par défaut, ou comme adresse source des paquets de découverte de son IP.

2.7 Déterminer rapidement le type d'une adresse

Outre les débuts de blocs souvent reconnaissables, des outils permettent de lever toute ambiguïté sur un type d'adresse.

Le logiciel `ipcalc` présent dans la plupart des distributions GNU/Linux offrait un couteau suisse précieux pour tous les administrateurs réseaux fainéants (et par conséquent compétents). Ils seront heureux de savoir qu'il existe un équivalent pour ces nouvelles adresses angoissantes, qui s'appelle logiquement `ipv6calc`.

Le fonctionnement est un peu moins optimal que son prédécesseur, mais offre beaucoup plus de possibilités de conversion.

Plutôt que de copier le `man` bien fourni, voici simplement un exemple :

```
$ ipv6calc -qi fe80::216:3eff:fed6:2ba0
Address type: unicast, link-local
Error getting registry string for IPv6 address: reserved(RFC4291#2.5.6)
Interface identifier: 0216:3eff:fed6:2ba0
EUI-48/MAC address: 00:16:3e:d6:2b:a0
MAC is a global unique one
MAC is an unicast one
OUI is: Xensource, Inc.
```

L'analyse d'une adresse de lien local permet ici de déterminer du même coup le type d'interface physique, en passant par l'adresse MAC (EUI-48). Dans ce cas précis, cette dernière est incluse dans l'adresse IPv6, ce point étant détaillé dans la section 4.2 page 37, dédiée à l'autoconfiguration *stateless*.

L'option `-i` permet de demander un affichage de tout ce que peut déduire le logiciel de cette adresse,

dont il détermine aussi le type automatiquement. L'option -q permet quant à elle de supprimer les lignes qui ne servent qu'à décrire le fonctionnement de l'outil.

Des options `--in` et `--out` permettent de faire un certain nombre de conversions (dont certaines ne semblent pas possibles actuellement), en précisant les types adéquats listés par les commandes :

```
$ ipv6calc --in -h
[...]
$ ipv6calc --out -h
[...]
```

Les tests permettront peut-être de révéler d'autres types d'adresses qui n'ont pas été référencés ci-dessus. Soit ils seront abordés dans les sections qui correspondent à leur utilité (principalement pour la cohabitation IPv4-IPv6), soit ils sont obsolètes (comme les adresses site-local unicast), soit ils font partie de solutions qui n'ont pas été retenues pour ce document.

2.8 Sélectionner l'IP de sortie

C'était déjà possible en version 4, mais avoir plusieurs adresses IP pour une même interface devient quelque chose de classique (voire systématique, dans le cas de l'autoconfiguration *stateless* avec les adresses de liens locaux, par exemple) en version 6.

Quelques commandes systèmes demandent de rajouter le nom de l'interface à utiliser lorsqu'elles demandent une adresse IP. Pour ce faire, il faut ajouter son nom à la fin de celle-ci, en le séparant par un symbole pourcent. Par exemple, pour une interface Unix :

```
2001:db8::abcd:def1:1234%eth0
```

Cette technique permet de forcer l'interface par laquelle sortiront les paquets, mais pas de sélectionner précisément l'adresse source qui sera utilisée. Il ne semble pas y avoir d'équivalent au pourcentage pour ça, et ce serait donc plutôt à la discrétion de la commande de fournir un paramètre pour la préciser (comme le `ping` de Windows, ou le `ping` étendu de Cisco).

Si l'adresse source n'est pas forcée, la RFC 3484 prévoit cet algorithme, qui sera appliqué par le système :

1. Si la destination est une adresse locale, il utilise celle-ci pour sortir.
2. Sinon il sélectionne l'adresse de plus petite portée, partagée avec celle de destination.
3. En cas d'égalité, il évite les adresses au format déprécié (dont la durée de vie conseillée est dépassée, mais pas la durée de validité).
4. Puis il privilégie les adresses de type « home »[9].
5. Il préfère ensuite une adresse de l'interface utilisée pour sortir.
6. Il continue en privilégiant le label correspondant[10].
7. Les adresses publiques sont utilisées en priorité sur celles restantes.

9. Cf. la section « *3.5. Mobility Addresses* » de la RFC 3484.
10. Cf. la règle 6 de la section « *5. Source Address Selection* » de la RFC 3484

8. Enfin, il choisit le plus long préfixe qui correspond.

Si, à la suite de toutes ces règles, il n'a pas encore trouvé de gagnant, il considère la dernière adresse utilisée.

Une solution pour inciter le système à mettre en retrait certaines adresses est de jouer sur la troisième règle. On notifiera donc qu'une adresse est dépréciée en forçant sa durée limite de validité à zéro (exemple pour GNU/Linux) :

```
# En l'ajoutant
# ip -6 addr add 2001:db8::abcd:def1:1234 dev eth0 preferred_lft 0
# En la modifiant
# ip -6 addr change 2001:db8::abcd:def1:1234/64 dev eth0 preferred_lft 0
```

2.9 Bonnes pratiques

Les bonnes pratiques (*best practices*) référencées dans cette liste s'inspirent des conseils répertoriés par Chris Grundemann [11], ARIN [12] (depuis la liste du NANOG) et mpreath [13] :

Préfixes de 64 bits pour les réseaux : Conformément à la RFC 4291, toutes les adresses unicast qui ne sont pas des liens d'interconnexion devraient utiliser un préfixe de 64 bits.

Utilisation du EUI-64 : Ces adresses devraient également utiliser l'identifiant d'interface (EUI-64 formé depuis l'EUI-48 correspondant à l'adresse MAC) pour former les 64 derniers bits, avec l'autoconfiguration *stateless*.

Préfixes /48 pour les sites : Un site (un bâtiment, un étage, campus, AS, interco, etc.) devrait utiliser systématiquement un /48.

Un /48 pour les adresses de l'infrastructure : Le premier ou le dernier /48 devrait être réservé pour l'infrastructure (boucles locales, interconnexions, etc.).

Adresses de loopback : Les adresses de boucle locale devraient utiliser un préfixe /128 [14], provenant du premier /64 du /48 réservé pour l'infrastructure.

Liens d'interconnexion : Malgré la recommandation d'utiliser les identifiants d'interface, un autre /64 du préfixe réservé à l'infrastructure devrait servir pour tailler des /127 destinés aux liens d'interconnexion. Comme décrit dans la section 5 de la RFC 6164, cette façon de faire évite le problème du ping-pong entre deux routeurs qui tentent de se renvoyer un paquet à destination d'une adresse de leur réseau mais qui ne correspond à aucune des deux adresses [15]. La seconde raison concerne le cache des *neighbors*, qui risque de se faire polluer par des adresses qui restent en cours de résolution, jusqu'à ce que la mémoire soit pleine (cette attaque n'est pas spécifique aux interconnexions,

11. http://www.ipbcop.org/wp-content/uploads/2012/02/BCOP-IPv6_Subnetting.pdf
12. http://www.getipv6.info/index.php/IPv6_Addressing_Plans
13. http://mattreath.com/2011/07/17/ipv6-best-practices
14. La RFC 4291 propose d'utiliser directement un /64 pour les liens d'interconnexion, puisque le /48 réservé pour l'infrastructure permet de toutes façons d'adresser 65536 /64.
15. La RFC 4443 sur l'ICMPv6 impose désormais aux routeurs de ne jamais faire suivre une route qui fait repartir un paquet par le lien par lequel il est arrivé. Ce problème existait déjà en IPv4, mais la rareté des adresses encourageait toujours les administrateurs à utiliser des /30 ou /31.

mais elles y sont particulièrement sensibles à cause du volume de leurs trafics). Enfin, certains préfèrent utiliser un `/64` pour obtenir des adresses courtes `x:x:x:x::1` et `x:x:x:x::2`.

Taille des préfixes égales par niveau : Chaque niveau de la hiérarchie de l'adressage devrait avoir le même préfixe. Cette recommandation semblait aussi naturelle en IPv4, mais était souvent contournée pour découper un maximum les préfixes afin d'adresser des sites plus petits que d'autres dans le but de faire des économies.

Numéro de VLAN dans les préfixes : Ajouter le numéro de VLAN dans l'adresse semble être une pratique courante [16] : pour un site qui a le préfixe `2001:db8:42::/48`, une machine dans le VLAN 201 aura pour préfixe `2001:db8:42:201`.

Adresses de lien local pour les routeurs : La RFC 4861 impose aux interfaces des routeurs de proposer une adresse de lien local, et celle-ci devrait toujours être utilisée pour les désigner lors de la création des routes sur le lien local (elles sont utilisées automatiquement lorsque c'est l'autoconfiguration *stateless* qui découvre la passerelle).

Adresse anycast des routeurs : Les interfaces des routeurs devraient utiliser l'adresse anycast correspondant au préfixe du réseau (l'ancienne « adresse du réseau », par exemple `2001:db8:201::/64`) pour permettre de les retrouver facilement.

Sous-réseaux par nibble : Les caractères hexadécimaux (*nibbles*) ne devraient jamais être à l'intersection entre les bits de réseau et d'hôte. En d'autres termes, le préfixe devrait toujours être un multiple de quatre, et les sous-réseaux devraient varier en fonction d'un seul caractère :

```
2001:db8:1000::/36
2001:db8:2000::/36
2001:db8:3000::/36
```

Plutôt que :

```
2001:db8:800::/37
2001:db8:1000::/37
2001:db8:1800::/37
```

Le RIPE met également à disposition un document [17] destiné à guider la conception d'un plan d'adressage IPv6.

Concernant la sécurité, un exemple de politique de sécurité pour un routeur GNU/Linux est donné en annexe A page 127. Elle vise à verrouiller un maximum les échanges, tout en respectant au maximum les différentes contraintes des RFC.

2.10 Jouer avec IPv6

Au delà des bonnes pratiques et du bon sens, la communauté des informaticiens a gardé son sempiternel sens de l'humour avec l'IPv6, qui permet de faire passer des messages à base d'un alphabet limité dans ses nouvelles adresses.

16. Le réseau Lothaire procède de cette façon.
17. http://www.ripe.net/lir-services/training/material/IPv6-for-LIRs-Training-Course/IPv6_addr_plan4.pdf/view

Le blog Coding Relic [18] propose une liste de mots anglais qui peuvent se glisser dans une adresse IPv6, la plupart utilisant les chiffres pour ajouter des lettres, à la façon du *leet speak* [19].

Citons les plus connus et intelligibles en anglais :

```
dead beef babe f00d caca b00b bad bed d00d
```

En français, le plus classique étant probablement `cafe:deca`, la liste des possibilités est moins intéressante (liste des mots de 1 à 8 lettres, de A à F plus les O remplacés par des zéros [20]) :

```
0de 0ff abbe acce:da acce:de acce:dee aede b0a b0b b0b0 b0f ba0b:ab baff:a baff:e
baff:ee bea bebe bec bee c0be:a c0c0 c0ca c0da c0de c0de:e c0fa:ce c0fa:cee ca
caca caca:0 caca:ba caca:be cade cafe ceda cede cede:e d0d0 dec dec0:da dec0:de
dec0:dee deca deca:de dece:da dece:de dece:dee ecaf:fa ecaf:fe ecaf:fee effa:ca
effa:ce effa:cee f0c faca:de fad0 fada fade fade:e fc0 fee
```

Certaines entreprises en profitent pour intégrer leur marque aux adresses de leurs serveurs :

```
$ dig AAAA +short facebook.com
2a03:2880:10:1f02:face:b00c:0:25
2a03:2880:10:8f01:face:b00c:0:25
2a03:2880:2110:3f01:face:b00c::
```

Enfin, on peut aussi y trouver des références culturelles [21] :

```
2001:db8::face:0f:b0e
```

Pour vérifier si une connexion utilise IPv6, le projet KAME [22] met à disposition sa célèbre tortue, accessible en HTTP et qui ne dansera que si elle est interrogée via une adresse en 128 bits.

18. http://codingrelic.geekhold.com/2011/04/ipv6-addresses-for-fun-and-profit.html
19. http://en.wikipedia.org/wiki/Leet
20. `for i in {1..8}/{a..f} {1..8}/o; do curl http://www.liste-de-mots.com/mots-nombre-lettre/$i/ | sed -n '/liste-mots/{n;s|</p>$||;:b;h;s|.*, ||;/^[a-fo0-9]\+$/{s|o|0|g;s|....|&:|;s|:$||;p};x;s|,[^,]\+$||;tb}'; done | unaccent UTF-8 | sort -u`
21. http://en.wikipedia.org/wiki/Face_of_Boe
22. http://www.kame.net

3

Compatibilité des systèmes

« *The Web does not just connect machines, it connects people.* » [1] - Tim Berners-Lee (2008)

3.1 GNU/Linux

3.1.1 Noyau Linux

Première apparition de l'IPv6 en 1996, pour le noyau 2.1.8, et disparition du statut d'expérimental en 2005 dans le noyau 2.6.12.

Les versions les plus récentes (2.6.x) supportent pleinement l'IPv6.

3.1.2 Debian / Ubuntu

L'IPv6 est activé par défaut et semble parfaitement fonctionnel pour un poste client, dès la version 4.0 (etch) de Debian en 2007. Côté Ubuntu, c'est la version 7.10 (Gutsy Gibbon) sortie la même année, qui active IPv6 par défaut. Dans les deux cas, le client NDP et DHCPv6 semble disponible.

Pour le support d'IPv6 au niveau des logiciels empaquetés, l'intégration complète de IPv6 étant un objectif (*release goal*[2]) de la version Squeeze de Debian, c'est la version minimale conseillée. Pour Ubuntu, la version 10.10 (Maverick Meerkat) semble être la première à particulièrement s'inquiéter de cet aspect.

IPv6 peut être désactivé en ajoutant la ligne suivante au fichier */etc/sysctl.conf* :

1. http://www.webfoundation.org/donations/knight2008/tbl-speech
2. http://wiki.debian.org/ReleaseGoals/FullIPv6Support

```
$ grep disable_ipv6 /etc/sysctl.conf
    net.ipv6.conf.all.disable_ipv6 = 1
```

Il faut donc vérifier son absence, et contrôler l'activation d'IPv6 en cherchant une éventuelle adresse locale :

```
$ ip a | grep inet6
    inet6~::1/128 scope host
    inet6 fe80::221:70ff:feec:b49b/64 scope link
```

Un `ping6` sur l'adresse locale peut aussi permettre de tester la bonne activation (sinon *connect : Network is unreachable*) :

```
$ ping6 ::1
    PING ::1(::1) 56 data bytes
    64 bytes from ::1: icmp_seq=1 ttl=64 time=0.044 ms
```

3.1.3 Fedora

L'IPv6 au niveau client semble parfaitement supporté à partir de la version 6 (Zod) de 2006, avec un client NDP et DHCPv6. Au niveau des paquets, la version 13 (Goddard) semble un minimum pour ne pas avoir de problèmes.

La vérification de la bonne activation de IPv6 peut se faire de la même façon que sous Debian. Pour l'activer, si ça n'est pas déjà fait, il faut modifier le principal fichier de paramètres du réseau, ainsi que celui concernant l'interface.

Pour */etc/sysconfig/network*, on peut ajouter (le *IPV6_DEFAULTGW*, avec une adresse réelle, est facultatif) :

```
NETWORKING_IPV6=Yes
IPV6_DEFAULTGW=2001:db8:2:657d:0:25:2:1234/64
```

Et pour */etc/sysconfig/network-scripts/ifcfg-eth0* (en adaptant le nom du fichier selon l'interface) :

```
IPV6INIT=yes
IPV6ADDR=2001:41D0:2:657d:0:25:2:1234
```

De la même façon que Debian, la plupart des outils d'administration réseau disposent de l'option `-6`.

3.2 Mac OS X

La première version qui semble supporter IPv6 correspond à la 10.2 (Jaguar) en 2002. Par contre, le NDP n'est disponible qu'à partir de la 10.3 (Panther) et le client DHCPv6 qu'à partir de la version 10.7

(Lion) - contrairement aux rumeurs[3]. Au niveau logiciels, c'est finalement la version 10.7 (Lion) qu'il faudra privilégier.

La configuration du réseau peut se faire graphiquement, aux mêmes emplacements que la configuration IPv4, avec la possibilité de passer la configuration IPv6 en manuelle ou automatique.

Pour désactiver l'IPv6, les versions avant Lion possèdent l'option « Éteint ». Sinon, la commande suivante est nécessaire :

```
# networksetup -setv6off Ethernet
# ou bien
# ip6 -x
```

3.3 Windows

3.3.1 Windows XP

Windows XP avec le service pack 2 (fin 2004) est la première version[4] à *supporter*[5] une version incomplète de l'IPv6 (notamment au niveau des tunnels, sur lesquels nous reviendrons), avec le support de NDP. Par contre il n'est pas activé par défaut, et il n'y a pas de client DHCPv6[6].

Pour activer l'IPv6 :

```
C:\> ipv6 install
Installation en cours...
Operation reussie.
```

Un simple `ipconfig` permet de valider la présence d'une adresse IPv6, et donc la bonne activation du protocole. Un `ping ::1` permet également cette vérification. Pour pinguer une machine du réseau, à l'instar de GNU/Linux, il est nécessaire de préciser l'interface à l'aide du symbole pourcent (`ping ipv6_distante%5`) si plusieurs cartes possèdent une adresse du même préfixe que l'IP à tester.

Pour afficher la liste des voisins connus, la commande `ipv6 nc` est disponible.

Pour ajouter une adresse à l'interface 5 :

```
C:\> ipv6 adu 5/fded:2092:eade:9f07:0:ffff:c0a8:5908
```

3. Cf. http://seclists.org/nanog/2011/Jul/417. La plupart des sites préviennent que Lion ne propose pas de DHCPv6 parce que ce choix ne fait pas partie des modes de configuration réseau proposés, comme il l'est en IPv4. Pour avoir confirmation qu'il est disponible, il suffit de choisir le mode automatique et de brancher un routeur qui stipule dans ses *Router Advertisements* qu'il faut utiliser un DHCP. L'expérimentation du DDNS section 8.8 page 116, utilise cette astuce.

4. http://ipv6int.net/systems/windows_xp-ipv6.html – À noter que Trumpet Winsock 5.0 (l'implémentation la plus connue à l'époque, de l'API TCP/IP standardisée) supportait déjà l'IPv6 alors qu'elle est apparue sous Windows 95.

5. http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/sag_ip_v6checklist.mspx?mfr=true

6. Il existe Dibbler, un client DHCPv6 libre qui peut être installé sous XP.

Pour pinguer une adresse de même préfixe, il faudra préciser l'adresse source pour indiquer l'interface à utiliser :

```
C:\> ping6 -s fded:2092:eade:9f07:0:ffff:c0a8:5908 fded:2092:eade:9f07:0:ffff:
    c0a8:593d
```

Une ligne *Microsoft TCP/IP version 6* est visible dans la fenêtre des *Propriétés de Connexion au réseau local*, mais ne permet pas de configurer les interfaces graphiquement (pas de bouton *Propriétés*).

3.3.2 Windows Vista / 7

Dès sa sortie en 2006 (pour les entreprises, il faudra attendre un an de plus pour les particuliers), Vista propose l'IPv6 activé par défaut, qui peut être configuré (ou désactivé) facilement depuis la fenêtre des paramètres réseaux. Par défaut, c'est l'autoconfiguration *stateless* avec NDP qui est active, mais un client DHCPv6 est disponible.

Concernant la configuration en ligne de commande, le commande `ipv6` de XP a été remplacée par `netsh interface ipv6`.

Par exemple, pour afficher les voisins connus :

```
C:\> netsh interface ipv6 show neighbors
```

3.4 Cisco

Un routeur Cisco active automatiquement le support de l'IPv6 dès lors qu'on lui indique une adresse.

La commande `ipv6 unicast-routing` permet d'autoriser le routage IPv6 entre les interfaces (équivalente à la commande `ip routing` de l'IPv4). La configuration minimale d'un routeur Cisco sorti d'usine ressemblera donc à :

```
Routeur> en
Routeur# conf t
Routeur(config)# ipv6 unicast-routing
Routeur(config)# int fa 0/0
Routeur(config-if)# ipv6 addr 2001:db8:a::1/64
Routeur(config-if)# no shut
Routeur(config-if)# int fa 0/1
Routeur(config-if)# ipv6 addr 2001:db8:b::1/64
Routeur(config-if)# no shut
```

De manière générale, la commande `ipv6` remplace la commande `ip`, ainsi pour obtenir des IP des interfaces :

```
Routeur# sh ipv6 int brief
FastEthernet0/0          [up/down]
    FE80::217:59FF:FE02:8DB8
    2001:DB8:A::1
```

```
FastEthernet0/1          [up/down]
    FE80::217:59FF:FE02:8DB9
    2001:DB8:B::1
```

La liste des routes s'obtient de la même façon qu'en IPv4 (`sh ipv6 route`). Mais alors qu'en IPv4 tous les administrateurs ont rêvé que IOS adopte la notation CIDR pour les définir, l'IPv6 ne laisse pas d'autre choix :

```
Routeur(config)# ipv6 route 2001:db8:c::/64 2001:db8:a::1
```

Certaines commandes apparaissent avec l'autoconfiguration *stateless*, et sont regroupées avec le mot-clé `nd` (*Neighbors Discovery*) au niveau de la configuration des interfaces :

- `ipv6 nd dad attempts <0-600>` : Force le nombre de tentatives (*Neighbor Solicitation*) que doit effectuer une interface avant de conclure qu'une absence de réponse (*Neighbor Advertisement*) signifie que l'adresse IP choisie est libre (cf. algorithme DAD section 4.2.3 page 39).
- `ipv6 nd managed-config-flag` : Positionne le drapeau qui permet d'indiquer dans les *Routeur Advertisement* (RA) que les adresses IP doivent se récupérer en DHCP (cf. tableau 4.1 page 45).
- `ipv6 nd other-config-flag` : Positionne l'autre drapeau des RA, qui permettra de prévenir que tout ce qui n'est pas adresse IP devra être réclamé au serveur DHCP (DNS, NTP, etc.).
- `ipv6 nd prefix X:X:X:X::X/<0-128>` : Permet de préciser le préfixe qui sera diffusé dans les annonces pour l'autoconfiguration *stateless* des nœuds du réseau (par défaut, c'est le préfixe de l'adresse de l'interface). Les durées maximales préférées et limites peuvent être ajoutées à la suite.
- `ipv6 nd ra interval <4-1800>` : Intervalle en secondes entre deux envois de RA non-sollicités par un nœud.
- `ipv6 nd ra suppress` : Le routeur n'enverra plus de RA régulièrement, et ne répondra plus aux *Routeur Solicitation*. Si un poste est en autoconfiguration *stateless* uniquement, il ne parviendra jamais à prendre une adresse IP, en l'absence de préfixe connu (cf. section 4.2 sur l'autoconfiguration *stateless* page 37).

Associer une adresse MAC avec une adresse IP ne se fait plus par la table ARP qui disparaît, mais par la table des voisins (les adresses MAC sont en notation pointée) :

```
Routeur# sh ipv6 neighbors
IPv6 Address                              Age Link-layer Addr State Interface
FE80::62FB:42FF:FEEF:E11A                   0 60fb.42ef.e11a REACH Fa0/0
2001:DB8:4503:105:62FB:42FF:FEEF:E11A       0 60fb.42ef.e11a REACH Fa0/0
```

Ces commandes sont pour IOS, mais peuvent différer pour les autres systèmes de Cisco (par exemple le `neighbor` du `sh ipv6 neighbor` de l'ASA ne prend pas de pluriel).

4

Autoconfiguration et DNS

« *On the Internet, nobody knows you're a dog.* » [1] - Peter Steiner

4.1 NDP et la récupération des adresses MAC sans ARP

4.1.1 Généralités

Le protocole ARP est banni de cette nouvelle version d'IP, au profit du protocole NDP (*Neighbor Discovery Protocol*, RFC 2461).

Plutôt que d'envoyer un *ARP Request* à l'ensemble des nœuds, la machine qui cherchera à associer une adresse MAC à une adresse IP utilisera le multicast pour envoyer en ICMPv6 un *Neighbor Solicitation* à une adresse de type *solicited-node* (figure 4.1 page 33).

104 bits	24 bits
`ff02::1:ff00:0`	24 derniers bits de l'adresse IPv6 cible

Figure 4.1 – *Format d'une adresse multicast* solicited-node.

Un exemple est fourni en figure 4.2.

L'adresse cible étant obligatoirement inscrite au groupe multicast correspondant à son adresse *solicited-node* [2], elle recevra cette réponse. Elle pourra donc retourner un message ICMPv6 de type *Neighbor*

1. http://www.unc.edu/depts/jomc/academics/dri/idog.html
2. Parler d'inscription pour une adresse multicast de lien local (`ff02:/16`) n'est pas tout à fait exact au niveau du routeur, puisque dans ce cas l'interface se contentera d'écouter les messages sans envoyer de message à celui-ci, qui relaiera sans se préoccuper de la composition du groupe.

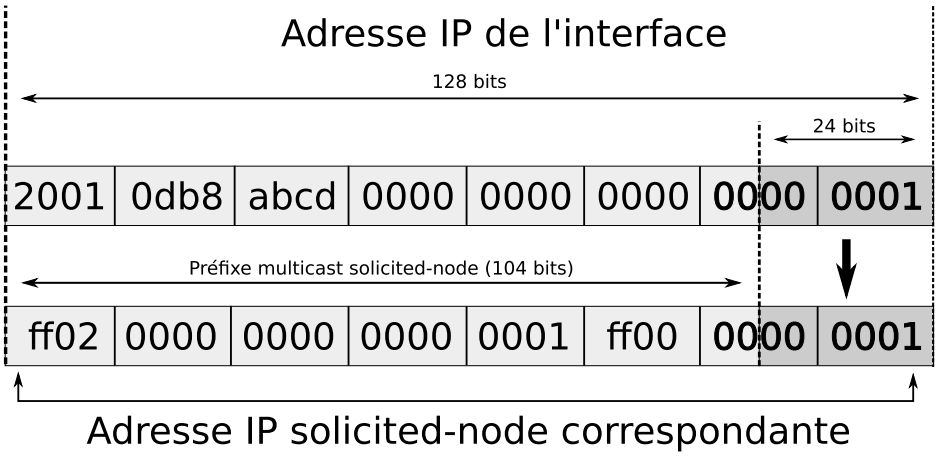


Figure 4.2 – *Conversion d'une adresse IP unicast en adresse multicast* solicited-node.

Advertisement, qui permettra au demandeur de déduire l'adresse MAC recherchée de l'adresse source de la trame et de compléter sa table des *neighbors* (équivalente à l'ancienne table ARP).

Le gain en terme d'encombrement des interfaces réseaux est providentiel : seules les quelques machines du lien qui ont en commun leurs 24 derniers bits interpréteront la trame, et seront à même de déterminer sa pertinence en fonction de l'adresse *who-has* contenue dans les données.

Puisque les adresses multicast de ce type ne sont pas routables, il convient de se méfier des problèmes qui peuvent être rencontrés lors du trajet retour. Ce problème n'est pas spécifique à IPv6, puisque les opérateurs de réseau qui routaient des adresses publiques IPv4 le rencontraient déjà et disposaient plus ou moins des mêmes solutions. Toutefois, la généralisation des NAT rendait ce problème invisible la plupart du temps. La nouveauté en IPv6 est donc qu'il se généralise jusqu'à impacter l'utilisateur lambda derrière son boîtier ADSL.

Ce problème est illustré dans les sections suivantes qui détaillent trois situations possibles, en s'attardant particulièrement sur les requêtes *Who has*, transportées en multicast via des messages ICMPv6 *Neighbor Solicitation*.

Le plan d'adressage choisi dans ces exemples est le suivant :

```
# Préfixe alloué par le responsable de R1
2001:0db8:abcd::/48

        # Intercos
        2001:0db8:abcd:1000::/52
                2001:0db8:abcd:1000::/56
                2001:0db8:abcd:1100::/56
                2001:0db8:abcd:1200::/56

        # Utilisateurs
        2001:0db8:abcd:2000::/52
                2001:0db8:abcd:2000::/56
                2001:0db8:abcd:2100::/56
```

4.1.2 Avec échanges de routes

En admettant que chaque routeur communique ses routes à celui qui le précède, l'envoi en multicast d'une requête *Who has* au poste utilisateur *préfixe* :2100::20 (U4) ne pose aucun problème puisque chaque routeur a conscience des sauts nécessaires et adresse donc les trames au routeur suivant plutôt que de directement chercher à utiliser l'adresse ethernet multicast correspondante.

Comme toutes les interfaces adressées en unicast, la machine U4 écoute en multicast pour le groupe correspondant à son adresse *solicited-node* calculée à partir de son adresse IP.

La règle du *more specific* s'appliquant, R1 utilisera la route *Directly connected* pour dialoguer avec R2 et considérera qu'il faut utiliser un saut pour toutes les machines hors du sous-réseau lui correspondant. La figure 4.3 illustre ce cas.

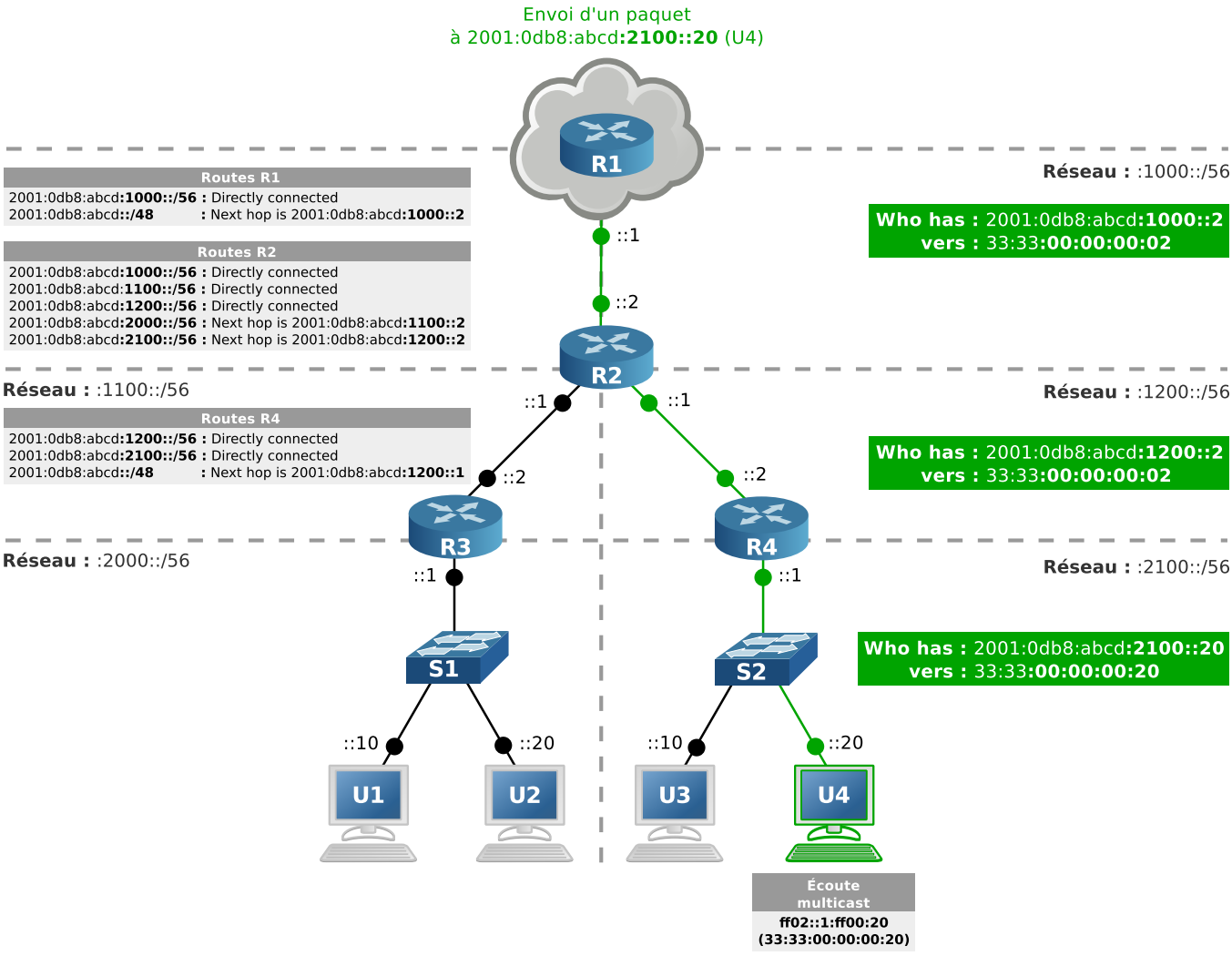


Figure 4.3 – *NDP avec diffusion des routes.*

4.1.3 Sans échange de routes

Dans le cas d'un FAI grand public positionné en R1, seul le préfixe est alloué et aucune route ne peut être échangée. Chaque fois que l'utilisateur :2100::20 souhaitera sortir vers Internet, R1 considérera qu'il est directement dans le même réseau que R2 puisqu'il n'a pas connaissance du redécoupage. Ainsi, il enverra directement la trame en utilisant l'adresse multicast en version ethernet.

Comme illustré en figure 4.4, puisqu'il s'agit d'un multicast de lien local, il n'y a pas d'inscription au groupe multicast auprès des routeurs. Ceux-ci communiquent donc les trames quoiqu'il arrive.

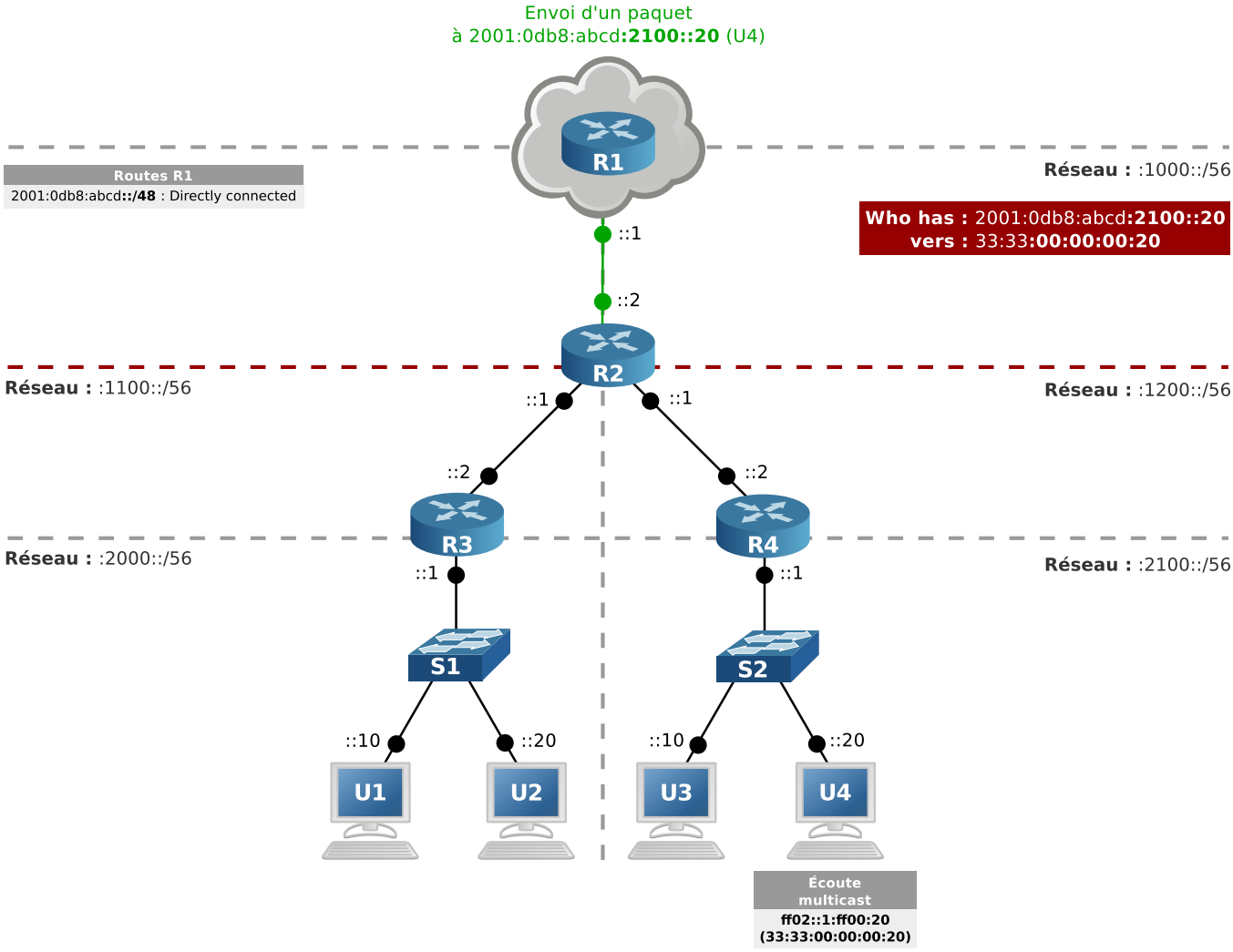


Figure 4.4 – *NDP sans diffusion des routes.*

4.1.4 Sans échange de routes avec proxies NDP

L'utilisation de proxies NDP (RFC 4389[3]) permet de contourner le problème de l'absence des routes, de façon similaire aux proxies ARP qui évitaient les NAT. Le routeur qui aura cette fonction écoutera tous les paquets en se mettant en mode *promiscuous* (ou au moins en mode *all-multicast*), et recevra donc

3. Avec des erreurs non-corrigées documentées ici : http://www.rfc-editor.org/errata_search.php?rfc=4389

la trame émise sur l'adresse ethernet multicast depuis le routeur du FAI, qu'il pourra relayer (figure 4.5 page 37).

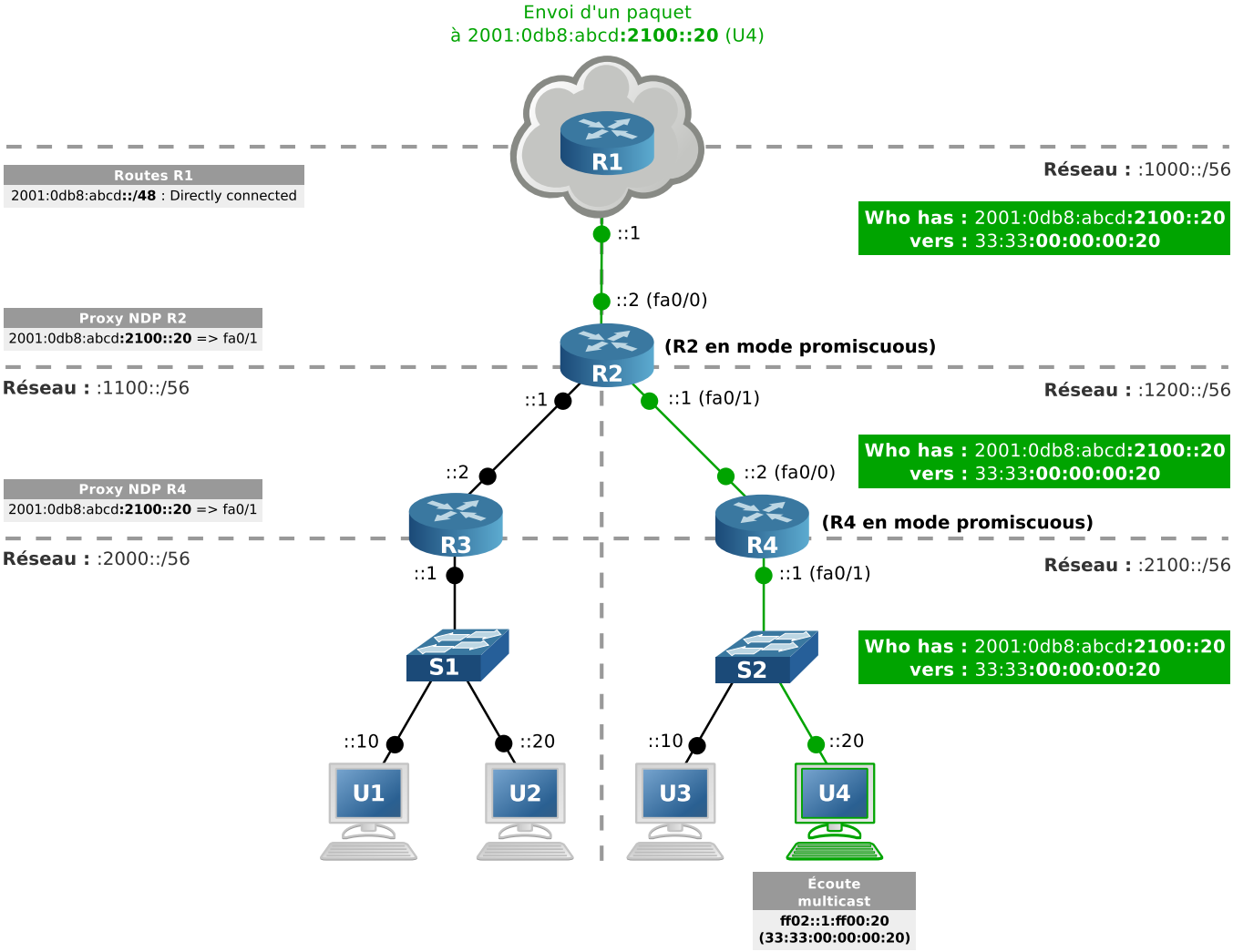


Figure 4.5 – *NDP sans diffusion des routes, avec l'utilisation d'un proxy.*

4.2 Autoconfiguration *stateless* (SLAAC)

4.2.1 Généralités

Toutes les version récentes des systèmes d'exploitation les plus populaires supportent l'autoconfiguration autonome (RFC 4862), en intégrant un client NDP. La méthode est souvent appelée « autoconfiguration *stateless* » ou sans état (ou SLAAC pour *Stateless Address AutoConfiguration*) en raison de l'absence de configuration au niveau du système.

La première étape pour une machine qui se configure de cette façon est de déterminer automatiquement son adresse IPv6. Elle utilise un préfixe de 64 bits auquel elle colle son adresse MAC (EUI-48) légèrement adaptée (EUI-64) qui sert d'identifiant d'interface (*interface/host ID*).

4.2.2 Construction automatique des adresses IP

Construction automatique de l'adresse IP :

1. Ajouter les octets `fffe` au milieu de l'adresse MAC de l'interface.
2. Positionner le septième bit de la MAC modifiée en partant de la gauche à 1 si l'adresse est unique (ce qui est le cas pour toutes les adresses MAC par défaut [4]) sinon 0.
3. Récupération du préfixe si c'est une adresse globale, sinon utilisation du préfixe d'adresse de lien local.
4. Concaténation du préfixe avec l'adresse MAC ainsi modifiée.

Le mécanisme est illustré en figure 4.6.

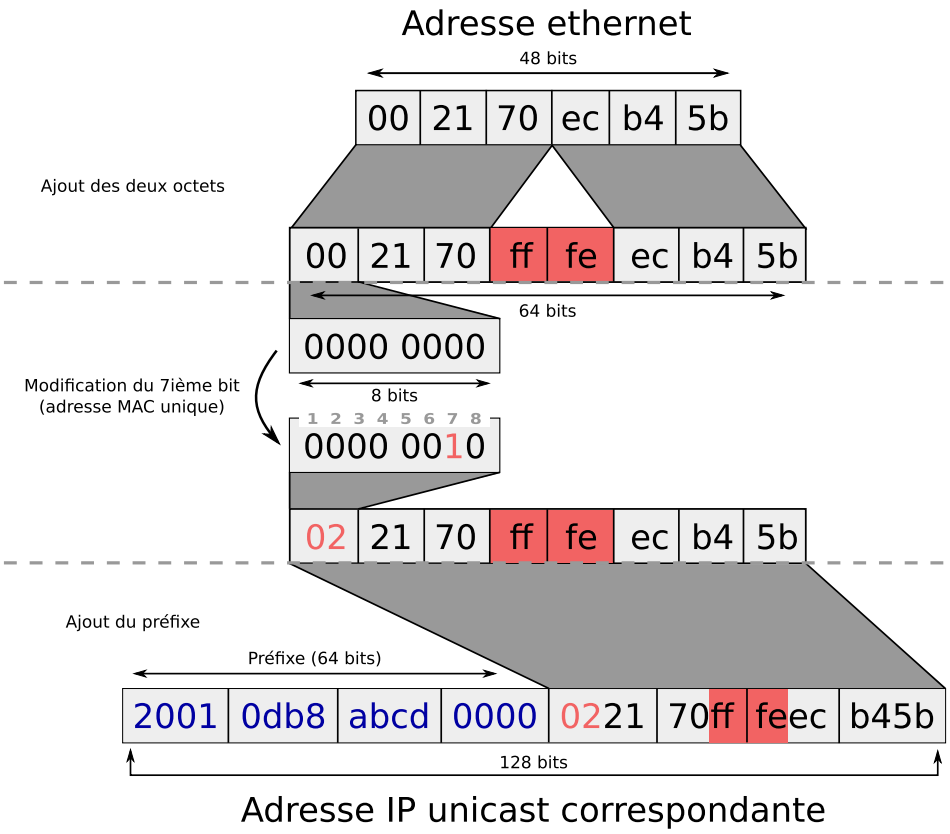


Figure 4.6 – *Autoconfiguration d'une adresse IP unicast avec l'autoconfiguration* stateless.

Le préfixe global s'obtient en écoutant les diffusions ICMPv6 des *Router Advertisement* sur le réseau. En général ce sont les routeurs qui se chargent de diffuser les préfixes toutes les cinq minutes, mais une machine a la possibilité de réclamer cette diffusion immédiatement en utilisant un message ICMPv6 de type *Router Solicitation* à destination de l'adresse *all-routers* (figure 4.7 page 39).

4. Ce septième bit correspond au bit U/L (*Universal/Local*) positionné à 1 si l'adresse n'est pas unique. L'inverse a été adopté dans cet algorithme pour faciliter l'administration des machines qui sera probablement manuelle si l'adresse ethernet a été changée manuellement (par exemple, on pourra donc utiliser l'adresse de lien local `fe80::1` plutôt que `fe80::200:0:0:1`). Si l'adresse MAC est bien formée, il suffira donc de simplement inverser la valeur de ce bit. Si l'adresse MAC n'est pas unique mais que son bit U/L est mal positionné, l'erreur se propagera dans l'adresse IPv6 calculée automatiquement.

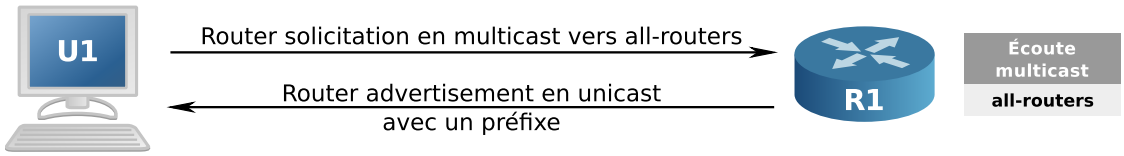


Figure 4.7 – *Diffusion des* Router Advertisements.

La RFC 6106 précise que les diffusions des routeurs peuvent diffuser une adresse DNS à utiliser par le même biais (RDNSS). Une interface configurée de cette façon garde toujours une adresse de lien local.

Pour activer la réception des messages RA des routeurs sous Mac OS avant Lion (ce qui peut aussi être fait en ajoutant `ip6mode=autohost` dans */etc/rc.conf*) :

```
1 # sysctl -w net.inet6.ip6.accept_rtadv=1
```

4.2.3 Duplicate Address Detection (DAD)

Afin de s'assurer que l'adresse choisie n'existe pas déjà sur le réseau, l'autoconfiguration *stateless* utilise le NDP avec des messages ICMPv6, de la même façon qu'il l'utilise pour remplacer l'ARP.

Elle commence par envoyer un message *Neighbor Solicitation* à l'adresse multicast *solicited-node* évoquée dans le tableau 2.2 page 18 (en utilisant l'adresse IP non-spécifiée composée de bits nuls comme adresse source). Pour pouvoir recevoir une éventuelle réponse, elle s'abonne à l'adresse multicast *all-nodes* de diffusion à tous les nœuds (le broadcast n'existant plus en IPv6). Si aucun nœud ne possède cette adresse, elle n'aura aucune réponse malgré ses essais répétés. Dans le cas contraire, elle recevra un message *Neighbor Advertisement* depuis l'IP en question vers l'adresse multicast de diffusion à tous les nœuds, qui confirmera la présence d'un conflit (illustration d'une collision en figure 4.8).

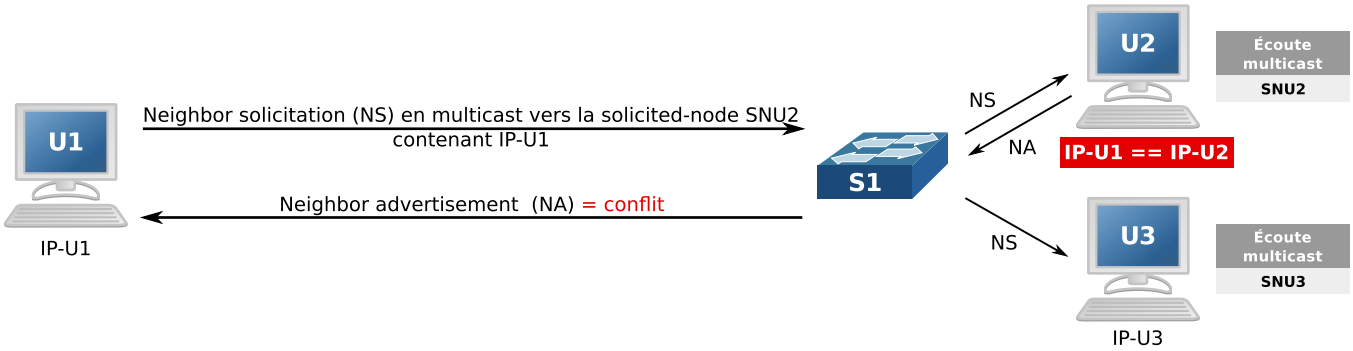


Figure 4.8 – *Détection d'une collision avec l'algorithme DAD.*

Une expérimentation de l'attribution automatique des adresses en mode *stateless* est proposée en section 8.2 page 89.

4.2.4 Durées de vie

Les messages *Router Advertisement* qui se chargent de fournir le préfixe à utiliser précisent toujours une durée de vie conseillée (*preferred lifetime*) et une durée de vie de validité (*valid lifetime*). Ce mécanisme remplace la notion de bail mise en œuvre par le DHCP.

La plupart du temps, la durée de vie conseillée n'expirera pas grâce aux annonces régulières des routeurs. Si ceux-ci ne viennent pas renouveler les baux, une adresse qui dépasse sa durée de vie conseillée devient dépréciée (*deprecated*). Les nouvelles sessions devront éviter ces adresses lors de leur sélection de l'IP de sortie (cf. règle 3 de l'algorithme décrit dans la section 4.2.3 page 39), et les sessions en cours continuent de l'utiliser. Dans le cas où c'est la durée de vie limite de validité qui est dépassée, l'adresse est supprimée de l'interface et toutes les sessions en cours qui l'utilisent sont fermées. Ce comportement est détaillé dans la RFC 4862.

Pour définir ces deux durées de vie sur un routeur Cisco (1000 secondes pour la durée de vie de validité et 900 secondes pour la durée de vie conseillée) :

```
1 Routeur(config)# ipv6 nd prefix 2001:db8::/64 1000 900
```

Ces durées de vie peuvent éventuellement être infinies en remplaçant ces deux valeurs par un `infinite`.

4.2.5 Problème de vie privée

La RFC 3041 décrit un problème de sécurité lié au calcul des adresses IP vu plus haut. Si une machine dérive chaque fois son identifiant d'interface depuis son adresse MAC qui reste constante, il devient aisé de le tracer de réseau en réseau en cherchant les 64 derniers bits identiques et les adresses deviennent pondérables. Un algorithme de hashage constitue une extension du protocole, pour assurer la préservation de la vie privée, qui doit prendre certains des mécanismes sus-mentionnés en considération.

Les objectifs de ce hashage sont donc les suivants :

1. Il doit prendre en considération l'algorithme de formation des adresses, c'est à dire conserver intact le préfixe global récupéré sur le réseau, et appliquer la modification du septième bit à gauche, selon l'unicité ou non de l'adresse MAC utilisée initialement.
2. Une adresse IP générée en autoconfiguration *stateless* doit toujours être associée à une date d'expiration, ce doit donc aussi être le cas pour sa version hashée.
3. Les adresses IP visibles quand la machine communique ne doivent pas être prévisibles, pour éviter que la procédure de hashage ne soit prédictible et ne permette de tracer la machine.
4. L'identifiant de l'interface doit être utilisé pour chacune des adresses de la machine. Dans le cas contraire, il faudrait s'abonner à toutes les adresses multicast de type *solicited-node* (pour recevoir les *Neighbor Solicitation*) correspondants à ces identifiants.

L'algorithme de hashage de la RFC suppose que la machine tient à jour une table de valeurs qui serviront à calculer les prochaines adresses. Afin que la machine puisse être contactée sur le réseau malgré ses adresses imprédictibles, celle-ci doit conserver son adresse dérivée de sa MAC en clair. Elle l'utilisera donc pour recevoir les connexions entrantes, et se servira de son équivalent hashé pour sortir.

Les adresses de sortie peuvent être calculée en utilisant les 64 derniers bits du *hash MD5* de la dernière valeur calculée (ou un nombre aléatoire pour la première itération), en tant qu'identifiant d'interface. Le septième bit en partant de la gauche doit être adapté pour être conforme à la convention. Les 64 derniers bits du *hash MD5* sont stockés en tant que nouvelle valeur, qui servira au prochain calcul d'identifiant. Cette méthode permet de s'affranchir des performances douteuses de la génération systématique de nombres aléatoires, en évitant de redemander un aléa à chaque itération et en utilisant à la base l'adresse MAC censée être unique.

Activer ce mécanisme pour Windows XP SP2/SP3 (pas d'IPv6 avant, et activation par défaut ensuite) :

```
C:\> netsh interface ipv6 set privacy state=enabled
```

Pour les Mac OS compatibles, avant 10.7 (Lion) qui l'active par défaut :

```
# sysctl -w net.inet6.ip6.use_tempaddr=1
```

Enfin côté GNU/Linux, si ça n'est pas déjà activé[5] :

```
# sysctl net.ipv6.conf.all.use_tempaddr=2
# sysctl net.ipv6.conf.default.use_tempaddr=2

# Relancer l'interface, ou attendre le prochain RA
# ifdown eth0
# ifup eth0
```

Pour Windows la modification sera permanente (ajouter `store=active` pour que ça soit temporaire). Pour GNU/Linux et Mac OS il faudra utiliser le fichier */etc/sysctl.conf* pour garder la modification au redémarrage.

Une méthode alternative de hashage destinée aux machines qui ne peuvent pas stocker de table de valeurs peut être employée, mais elle dépasse le cadre de ce document destiné à des machines clientes standards.

4.2.6 Inconvénients

L'autoconfiguration *stateless* convient parfaitement aux petits réseaux, qui peuvent ainsi se passer totalement de configuration de leurs machines pour les faire communiquer entre elles. Toutefois, elle devient problématique dès lors que le réseau nécessite de pouvoir associer une adresse à une machine. Un mécanisme de conservation de journaux systèmes pour associer les adresses aléatoires avec les adresses contenant les adresses MAC en clair anéantirait la possibilité alléchante de se passer des contraintes à ce niveau, autrefois induites par l'utilisation des NAT.

De plus, l'utilisation d'une adresse aléatoire en sortie ne permet pas de lui associer un champ `PTR` systématique au niveau du DNS, ce qui incite certains serveurs exigeants à les considérer plus facilement comme malveillantes.

Au niveau de la sécurité, laisser les machines s'arranger entre elles n'est pas anodin. De nombreuses

5. https://bugs.launchpad.net/ubuntu/+source/procps/+bug/176125

attaques de type *redirect*, *spoofing* ou *flooding*[6] sont décrites dans la RFC 3756, véritable bible du petit pirate de l'autoconfiguration *stateless*. Certaines de ces attaques pourraient être évitées en utilisant le mécanisme SEND (*SEcure Neighbor Discovery*, RFC 3971/3972), qui impose aux machines de se calculer des adresses dites CGA (*Cryptographically Generated Address*) à l'aide de leur adresse calculée en clair et d'une clé publique (RFC 3972). Ce mécanisme est prévu par les messages de sollicitation et annonces de voisins, qui peuvent embarquer les clés publiques. Cette solution entache la simplicité de ce type d'autoconfiguration.

Une solution plus simple[7] que SEND consiste à définir une ACL au niveau du commutateur principal, qui s'appliquera sur tous les ports sauf ceux légitimes à envoyer des RA (le 137 correspondant aux *Redirect Message* qui sont aussi contrôlés au passage) :

```
Switch(config)# ipv6 access-list RA-snooping
Switch(config-ipv6-acl)# deny icmp any any router-advertisement any
Switch(config-ipv6-acl)# deny icmp any any 137 any
Switch(config-ipv6-acl)# permit ipv6 any any
```

Toutefois, cette solution ne résout pas tous les problèmes, et nécessite des commutateurs suffisamment évolués pour supporter l'utilisation des ACL.

Pour toutes ces raisons, un réseau qui nécessite une administration massive avec un souci de traçabilité de l'activité, tend à impliquer de préférer la seconde méthode d'autoconfiguration, plus traditionnelle. Dans tous les cas, désactiver l'autoconfiguration *stateless* sur les serveurs et autres machines qui ne sont pas concernées par ce mode de configuration semble une saine initiative.

Voir également les règles de pare-feu conseillées dans la section 2.9 page 24.

4.2.7 Désactiver

Sous GNU/Linux, l'autoconfiguration *stateless* peut être désactivée avec les commandes suivantes :

```
# sysctl -w net.ipv6.conf.all.accept_ra_defrtr=0
# sysctl -w net.ipv6.conf.default.accept_ra_defrtr=0
# sysctl -w net.ipv6.conf.all.autoconf=0
# sysctl -w net.ipv6.conf.default.autoconf=0
```

L'autoconfiguration utilise les *Router Advertisements* (RA) pour récupérer les paramètres. Cependant, la version `autoconf` ne concerne que l'attribution automatique des adresses formées grâce à la distribution des préfixes. Alors que la version `ra` se charge spécifiquement de découvrir et positionner toute seule la route par défaut en trouvant l'adresse de lien local du routeur, avec en plus une route par préfixe trouvée sur l'ensemble des RA reçus (découverte automatique des passerelles).

Chaque fois qu'un RA est reçu, les temps d'expiration des adresses autoconfigurées sont remis à leur maximum indiqué par le RA, uniquement pour les préfixes annoncés à ce moment là. Si un préfixe n'est plus annoncé, il finira par expirer, mais ne sera pas supprimé pour autant au prochain RA.

6. Une vidéo éloquante de l'arrêt d'un Windows avec un simple *flood* de RA, avec les explications (qui concernent aussi les *BSD), est disponible ici : http://samsclass.info/ipv6/proj/flood-router6a.htm.
7. Merci à Romain BOISSAT (http://chroot-me.in).

Pour désactiver temporairement les adresses obtenues par autoconfiguration :

```
# ip -6 addr flush dynamic
```

Deux acteurs externes se font souvent remarquer lorsque l'administrateur souhaite reprendre le contrôle complet de ses interfaces :

- **Le mDNS :** il faut alors stopper le service *avahi-daemon*.
- **Les gestionnaires de réseaux :** il faut bien penser à stopper aussi tout ce qui est *wicd*, *NetworkManager* (pour GNU/Linux), etc. Ces démons ont tendance à supprimer régulièrement tout ce qu'ils n'ont pas configuré eux-mêmes, sur toutes les interfaces dont ils ont connaissance (ou *a minima* lorsqu'elles sont relancées logiciellement). À noter que certaines versions de NetworkManager (dans Ubuntu 12.04 par exemple), nécessitent de cocher son option *Ignorer les routes obtenues automatiquement*, sous peine sinon de le voir ajouter une route par destination contactée.

4.3 Autoconfiguration *stateful* (DHCPv6)

4.3.1 Généralités

Cette méthode d'autoconfiguration utilise un serveur DHCPv6, de façon très similaire au protocole DHCPv4, moyennant un léger reconditionnement des messages échangés et l'utilisation des adresses anycast (RFC 3315). Par exemple, pour recenser la liste des serveurs DHCP du réseau, le client enverra un message de type *SOLICIT* (correspondant à l'ancien *DHCPDISCOVER*) à l'adresse anycast du routeur du sous-réseau (comme indiqué lors de la présentation du type anycast), qui renverra un message de type *ADVERTISE* (anciennement *DHCPOFFER*).

Comme dans sa version 4, le DHCP pourra retourner un domaine ainsi que des adresses DNS. La mise en place de relais est toujours possible.

Une nouveauté intéressante se situe du côté de la renumérotation des réseaux, qui peut s'avérer nécessaire dans le cas d'un changement de FAI, par exemple. Le serveur a la possibilité de diffuser en multicast un message de type *RECONFIGURE*, qui incitera tous les clients à lui envoyer une réponse de type *RENEW* (ou *INFORMATION REQUEST*, selon la configuration du serveur) pour acquérir les informations à jour. L'effet du changement est alors immédiat.

Les différents messages utilisés par cette nouvelle version de DHCP sont décrits dans la section suivante.

4.3.2 Protocole

Les messages DHCP (envoyés en ICMPv6) qui existaient dans DHCPv4 changent de nom (l'ancien nom est rappelé entre parenthèses), et des nouveautés sont ajoutées :

SOLICIT (DHCPDISCOVER) : Permet au client de faire un appel général sur son réseau pour localiser les serveurs DHCPv6. Le message est envoyé en multicast à l'adresse `ff02::1:2` (RFC

3315).

ADVERTISE (DHCPOFFER) : Réponse des serveurs DHCPv6 aux *SOLICIT*, en unicast.

REQUEST (DHCPREQUEST) : Utilisé par le client pour demander les paramètres de configuration au serveur DHCPv6 sélectionné.

CONFIRM : C'est une nouveauté, et elle permet au client de simplement s'assurer auprès du serveur que son ou ses adresses sont toujours valides.

RENEW (DHCPREQUEST) : Permet au client de demander un prolongement du bail, ou mettre à jour ses paramètres si ceux-ci ont changés depuis.

REBIND (DHCPREQUEST) : Ce message a la même fonctionnalité que le *RENEW*, mais il est utilisé pour interroger en multicast l'ensemble des serveurs DHCPv6, lorsque le serveur DHCP d'origine ne répond plus.

RELEASE (DHCPRELEASE) : Envoyé par le client au serveur pour indiquer à ce dernier la valeur des paramètres actuellement utilisés.

DECLINE (DHCPDECLINE) : Permet au client de signifier au serveur que les paramètres transmis ne peuvent pas être utilisés.

REPLY (DHCPACK et DHCPNACK) : Il s'agit du message contenant la réponse du serveur DHCP pour les deux types d'interrogations *RENEW* et *REBIND*, ou pour confirmer la bonne réception des deux types de messages précédents.

INFORMATION-REQUEST (DHCPINFORM) : Permet simplement au client de demander de nouveaux paramètres de configuration.

RECONFIGURE (DHCPFORCERENEW) : Permet au serveur d'indiquer à ses clients que les paramètres doivent être actualisés, et qu'il serait bon qu'ils le sollicitent avec un message *RENEW* ou *INFORMATION-REQUEST*.

RELAY-FORWARD : C'est une seconde nouveauté, et c'est utilisé par les relais pour transmettre à un serveur (ou un autre relais) le message initial du client, qui sera contenu dans les options.

RELAY-REPLY : Il s'agit du pendant du type précédent, permettant de répondre à la question du client initial au travers d'un relais.

Concernant les relais DHCPv6 en particulier, le fonctionnement diffère un peu de celui de son ancêtre (figure 4.9 page 45).

4.3.3 Compatibilité des clients

Contrairement au NDP utilisé par l'autoconfiguration *stateless* et installé par défaut sur tous les systèmes récents, le mode *stateful* nécessite un client DHCP qui n'est pas toujours présent sur les machines.

Côté Windows, si on exclut la version XP qui ne dispose pas de client par défaut (mais sur lequel on peut installer un client libre[8]), les versions suivantes ne nécessitent pas d'installation supplémentaire.

Des drapeaux `M` (*ManagedFlag*) et `O` (*OtherConfiguration*) sont à la disposition de l'administrateur pour trouver des compromis entre l'autoconfiguration *stateless* et *stateful* (qu'on appellera DHCPv6

8. Dibbler : http://klub.com.pl/dhcpv6/#DOWNLOAD

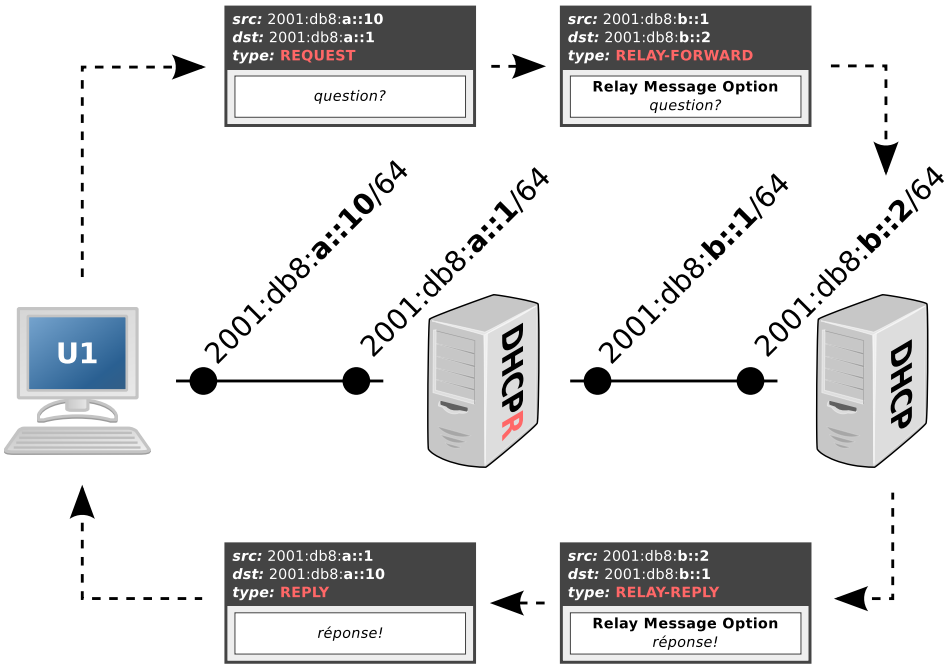


Figure 4.9 – *Fonctionnement d'un relais DHCPv6.*

stateless, de la RFC 3736) et peuvent être combinés comme indiqué dans le tableau 4.1 (la dernière configuration est un peu étrange et n'a pas beaucoup d'intérêt).

O	M	SLAAC (avec RDNSS)	DHCPv6
0	0	Gère tout	Désactivé
1	1	Désactivé	Gère tout
1	0	Gère les IP	Gère le reste dont les DNS
0	1	Gère le reste dont les DNS	Gère les IP

Table 4.1 – *Signification des drapeaux de répartition des tâches pour l'autoconfiguration.*

Ces drapeaux peuvent être positionnés avec la commande suivante (versions supérieures à XP), *managed* correspondant au flag `M` et *advertise* au flag `O` :

```
C:\> netsh interface ipv6 set interface nom_interface managed=enable advertise=
   disable
```

Côté GNU/Linux, le client est disponible directement dans les paquets, et s'appelle en général *isc-dhcp-client* (pour la version standard, sinon Dibbler, disponible aussi dans les paquets, semble aussi très usité) pour les distributions basées sur Debian. La version ISC (*Internet Systems Consortium*) ne semble pas permettre d'affiner la configuration des drapeaux comme sous Windows.

La version Dibbler les propose, via le fichier de configuration du client :

```
iface nom_interface {
 stateless
 option dns-server
 option domain
```

```
}
```

Une expérimentation du DHCPv6 statique est disponible à la section 8.7 page 109. Le DHCPv6 dynamique est utilisé lors de l'expérimentation de la section 8.8 page 116.

4.3.4 Configuration des serveurs

Le partage entre le SLAAC et le DHCPv6 se fait aussi côté routeur. Pour un routeur Cisco, on utilisera les options *managed-config-flag* et *other-config-flag* :

```
Routeur(config)# int fa 0/0
Routeur(config-if)# ipv6 nd managed-config-flag
Routeur(config-if)# ipv6 nd other-config-flag
```

Par exemple, pour déléguer la configuration des adresses au mode *stateless* mais fournir les serveurs DNS en DHCPv6 (ce qui semble obligatoire tant que le RDNSS ne sera pas mieux supporté) :

```
Routeur(config)# ipv6 dhcp pool DNSPool
Routeur(config-dhcp)# dns-server 2001:db8::1
Routeur(config-dhcp)# domain-name u1.example
Routeur(config-dhcp)# int ethernet 0/0
Routeur(config-if)# ipv6 address 2001:db8::1/64
Routeur(config-if)# ipv6 enable
Routeur(config-if)# ipv6 nd other-config-flag
Routeur(config-if)# ipv6 dhcp server DNSPool
```

La même chose peut-être obtenue en configurant un relais plutôt que de laisser l'infâme DHCP de l'IOS faire le boulot :

```
Routeur(config-if)# ipv6 dhcp relay destination 2001:db8::2
```

Si radvd est utilisé pour le *stateless* sur un GNU/Linux :

```
interface eth0
{
  AdvManagedFlag off;
  AdvOtherConfigFlag on;
};
```

Lorsqu'une plage d'IPv6 est indiquée au serveur DHCP pour indiquer les adresses disponibles, il ne faut pas oublier que `::10` ne suit pas `::9`, mais qu'il y a `::a`, `::b`, `::c`, `::d`, `::e` et `::f` entre les deux.

4.3.5 Adresses statiques (DUID)

Plutôt que d'utiliser systématiquement les adresses MAC pour identifier les machines, le DHCPv6 se concentre sur les DUID (*DHCP Unique Identifier* de la section 9 de la RFC 3315).

Les deux principaux types de DUID sont DUID-LL (*Link-Layer*) et DUID-LLT (*Link-Layer plus Time*). Comme leur nom l'indique, les deux se basent sur l'adresse MAC (dans le cas d'une utilisation d'ethernet) mais le second ajoute un *timestamp* dans la construction, comme illustré en figure 4.10.

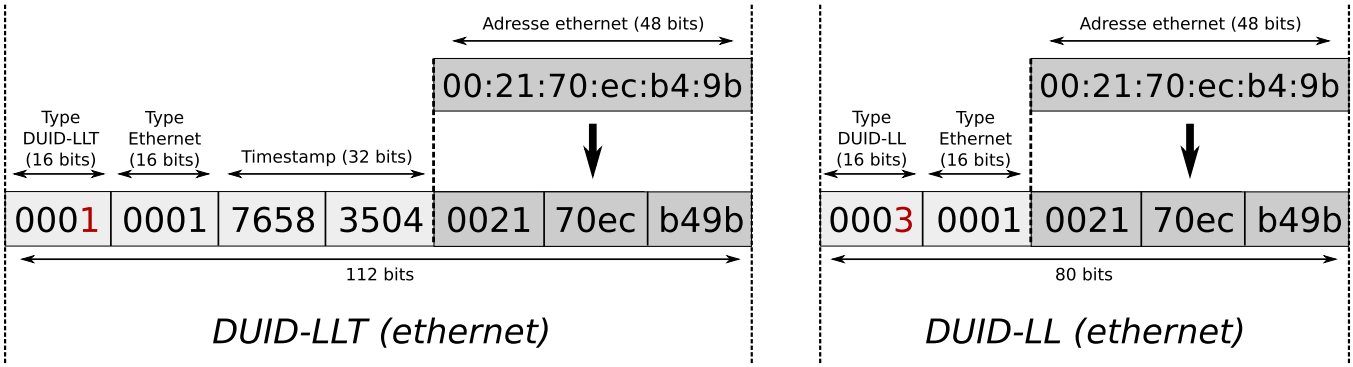


Figure 4.10 – *Confection d'un identifiant DHCP de type DUID.*

Le DUID-LLT est celui utilisé par défaut par les clients DHCP de la version ISC. L'identifiant est calculé une première fois avec le *timestamp* courant et stocké dans le fichier de baux (*/var/lib/dhcp/dhclient.leases6*). Si celui-ci est régénéré pour une raison ou une autre, l'identifiant change, et les tables d'association DUID-IP sur les serveur DHCP ne sont donc plus valides. Le DUID utilisé par le client est visible dans les paquets DHCPv6 scannés par les *sniffers*.

Ainsi, il vaut mieux préférer une version totalement statique (s'il existe déjà, penser à supprimer le fichier de baux */var/lib/dhcp/dhclient6.leases* auparavant, sinon il s'évertuera à utiliser le DUID-LLT calculé lors des sessions précédentes) :

```
1 # dhclient -D LL eth0
```

Cette contrainte complique l'utilisation du DHCP statique en IPv6.

Le DHCPv6 statique ne semble pas encore supporté sur les équipements Cisco, mais est très bien supporté sur la version serveur de l'ISC. Un exemple de configuration (avec des relais DCHP) est donné pour l'expérimentation de la mobilité IPv6 en section 8.7 page 109.

4.4 Résolutions DNS

4.4.1 Généralités

Puisque les adresses IPv6 sont quatre fois plus longues que les adresses IPv4, ce sont les champs DNS `AAAA` qui contiendront ces adresses. Un même domaine peut faire cohabiter une réponse pour `A` et pour `AAAA`, et ce sont les requêtes qui décideront ce qu'elles demandent.

Exemple de double réponse :

```
1 $ host ipv6actnow.org
2 ipv6actnow.org has address 193.0.19.50
3 ipv6actnow.org has IPv6 address 2001:67c:2e8:11::c100:1332
```

Exemple de résolution inverse (nouveau domaine `ip6.arpa`, qui remplace le `ip6.int`) :

```
$ host 2001:67c:2e8:11::c100:1332
2.3.3.1.0.0.1.c.0.0.0.0.0.0.0.0.1.1.0.0.8.e.2.0.c.7.6.0.1.0.0.2.ip6.arpa domain
    name pointer buzzard.ipv6.ripe.net.
```

Calculer un `PTR` avec `ipv6calc` :

```
$ ipv6calc --in ipv6 --out revnibbles.arpa 2001:67c:2e8:11::c100:1332
2.3.3.1.0.0.1.c.0.0.0.0.0.0.0.0.1.1.0.0.8.e.2.0.c.7.6.0.1.0.0.2.ip6.arpa.
```

L'IANA a ajouté des enregistrements `AAAA` à six *root-servers* en 2008, rendant possible la résolution de noms de domaine entièrement en IPv6.

Des enregistrements A6 sont parfois cités dans la littérature. Alors que la documentation de IBM [9] qualifie les `AAAA` de « *Obsolete format of IPv6 address* », un mémo [10] (subjectif) sur le site de l'IETF conseille d'oublier les champs A6 (« *move A6 document to historic state* »), principalement pour des raisons de sécurité. Le principal apport des champs A6 consiste à pouvoir fragmenter les requêtes pour obtenir des fichiers de zones plus simples à gérer (voir le mémo), au détriment de la quantité de travail pour les *resolvers*. Le déploiement actuel de l'IPv6 semble de toutes façons avoir largement donné raison aux « quadras » (AAAA).

4.4.2 DHCPv6

À l'instar de son prédécesseur, le DHCPv6 est habilité à distribuer des adresses de serveurs DNS. Son fonctionnement est détaillé dans la section 4.3 page 43.

Tous les systèmes ne proposent pas de client DHCPv6 par défaut :

- Des clients DHCPv6 pour GNU/Linux sont disponibles la plupart du temps dans les dépôts officiels : *isc-dhcp-client* ou *dibbler-client*.
- Chez Mac OS X, il est disponible par défaut depuis la version 10.7 (Lion).
- Pour Windows, il est aussi disponible par défaut depuis Vista.
- Enfin pour Cisco le *Features Navigator* indique qu'il serait disponible depuis la version 15.2(2)S de IOS.

4.4.3 RDNSS

Les options RNDSS (*Recursive DNS Server*) et DNSSL (*DNS Search List*) sont des ajouts récents du protocole NDP. La RFC 6106 qui les documente explique que tant que les machines étaient en double pile, elles pouvaient utiliser les DNS en IPv4. Mais dès lors que la machine est entièrement IPv6, un DHCPv6 était nécessaire pour diffuser les adresses de DNS, ramenant le protocole NDP au niveau de la futilité étant donné l'importance de ces adresses pour l'autoconfiguration des postes clients.

9. http://pic.dhe.ibm.com/infocenter/aix/v7r1/index.jsp?topic=%2Fcom.ibm.aix.files%2Fdoc%2Faixfiles%2Fnamed.conf.htm
10. http://tools.ietf.org/html/draft-ietf-dnsext-aaaa-a6-01

Le RNDSS permet de diffuser les adresses des serveurs de nom de domaine via les annonces de l'autoconfiguration *stateless*, tandis que le DNSSL permet de diffuser la liste des suffixes des domaines à utiliser par défaut sur le réseau.

Puisque ces options sont récentes, elles ne sont pas supportées partout :

- Le monde GNU/Linux semble le plus en avance sur ce point, puisque la plupart des versions récentes des distributions supportent le RDNSS par défaut.
- Comme pour le client DHCPv6, Mac OS X ne supporte cette option par défaut que depuis la version 10.7 (Lion).
- Du côté du monde Windows le constat est plus pessimiste puisqu'il n'y a pas de support natif y compris sur Seven, mais uniquement le logiciel libre *rdnssd-win32* à installer.
- Au niveau serveur chez Cisco, cette option n'est pas encore implémentée.

Un serveur DNS récursif est un serveur DNS qui déléguera la question à un autre serveur DNS pour retransmettre la réponse, lorsqu'il ne la connait pas lui-même. Il s'agit du type de serveur DNS que tous les clients finaux utilisent, et qui s'oppose aux serveurs DNS uniquement destinés à faire autorité sur un domaine, sans pour autant être apte à répondre à toutes les requêtes d'une machine (évitant ainsi un cas fréquent de déni de service). Il s'agit donc du même type d'adresse de serveur DNS qu'un serveur DHCP est habitué à distribuer.

L'autoconfiguration *stateless* entièrement en IPv6 n'aura aucun sens tant que cette option ne sera pas mieux supportée par les systèmes. Des possibilités de mixage de DHCPv6 et RDNSS sont disponibles dans la section 4.3 page 43. L'expérimentation à la section 8.2 page 89 prouve que le RDNSS est malgré tout très bien supporté sous GNU/Linux.

4.4.4 mDNS

Le DNS multicast consiste à interroger le réseau plutôt qu'un serveur DNS désigné pour résoudre des noms de domaine. Ce mécanisme fait partie des outils *zéroconf*[11]. Il n'est pas une nouveauté d'IPv6, mais peut constituer une piste à explorer pour la distribution des adresses DNS.

Les deux principales spécifications existantes sont LLMNR[12] (Microsoft) et le couple mDNS/DNS-SD[13] (Apple). Il existe une troisième spécification SLP (Hewlett-Packard) peu utilisée, mais qui a le mérite d'être la seule à avoir réussi à faire l'objet d'un consensus suffisant pour aboutir sur une normalisation de l'IETF (RFC 2608 et RFC 3224).

Alors que LLMNR n'est utilisé que sur Windows, mDNS/DNS-SD est largement utilisé sous Mac OS X avec l'implémentation Bonjour/Rendezvous et est disponible sous GNU/Linux avec Avahi (qui est activé par défaut dans quasiment toutes les distributions).

L'utilisation classique du mDNS est de rechercher une machine sur le lien local à partir de son nom plutôt que de son IP, en lui ajoutant le suffixe *.local* réservé à cet usage. Ainsi, pour une nouvelle imprimante simplement branchée directement sur une machine ou via un commutateur, l'utilisateur pourra se contenter de saisir *myprinter.local* dans son navigateur pour obtenir la page web de son

11. http://www.zeroconf.org
12. RFC informational 4795
13. http://tools.ietf.org/html/draft-cheshire-dnsext-multicastdns-06

nouveau périphérique. Concrètement, en utilisant le suffixe du mDNS, le système a interrogé le réseau en demandant qui était *myprinter* à l'adresse multicast `ff02::fb`, en UDP sur le port 5353. La première machine à écouter cette adresse de diffusion qui se reconnait et qui répond en unicast, répond à la question grâce à l'adresse source de sa réponse.

Cette fonctionnalité est disponible par défaut sur quasiment tous les systèmes, qui écoutent donc tous par défaut l'adresse multicast mDNS (comme en témoigne par exemple la section 2.3.5 page 19 qui analyse les abonnements par défaut d'une Debian).

Il est possible d'étendre cette fonctionnalité à des résolutions plus larges (récursives) de noms de domaine, de façon à se passer complètement de serveur DNS. Cette possibilité n'est jamais activée par défaut puisque c'est un trou de sécurité béant : la première réponse arrivant du réseau fait foi quelque soit sa provenance (risque de *spoofing/fishing*).

À titre de documentation, voici comment activer ce mode de résolution sous Debian [14] (paquet *nss-mdns*) :

1. Vérifier que `mdns` est bien présent sur la ligne `hosts` de */etc/nsswitch.conf* (c'est le cas par défaut).
2. Ajouter `search local` dans */etc/resolv.conf*.
3. Activer la résolution des noms de domaine en multicast, en créant le fichier */etc/mdns.allow* avec simplement le caractère * dedans.

4.4.5 Résolutions inverses des adresses autoconfigurées (DDNS)

Une pratique courante pour lutter contre le *spamming* ou le *flooding* consiste à vérifier systématiquement si l'IP qui cherche à contacter le service (par exemple un SMTP) possède bien un champ `PTR`. Certains vont même jusqu'à vérifier ensuite que le domaine retourné correspond bien à l'IP initiale. Alors que les serveurs sont souvent adressés statiquement, cette problématique se pose surtout pour les réseaux wifi librement accessibles. Bien que la problématique ne soit pas nouvelle, elle est à nouveau naturellement amplifiée par l'absence de NAT, qui ne nécessitait que l'enregistrement des quelques IPv4 publiques qui étaient utilisées pour sortir.

En mode *stateful* (DHCPv6), le DHCP peut communiquer avec un serveur DNS pour ajouter automatiquement un domaine et un reverse chaque fois qu'une IPv6 est allouée. Comme en IPv4, il s'agit du DDNS (*Dynamic DNS*, RFC 2136 et 3007) présenté en figure 4.11.

Une expérimentation est réalisée dans la section 8.8 page 116.

Pour le mode *stateless*, la même manipulation est réalisable. Mais puisque chaque machine s'attribue elle-même une adresse IP, elle doit être elle-même autorisée à ajouter dynamiquement des entrées dans le serveur DNS. Chacune d'entre elle devra donc posséder une clé de chiffrement limitée et référencée auprès du serveur DNS. La manipulation est donc bien réalisable mais pas vraiment envisageable. Ce nouveau problème, additionné à celui du RDNSS qui n'est pas encore exploitable, met l'autoconfiguration *stateless* dans une situation bien inconfortable face aux DNS.

Citons tout de même quelques projets qui tentent de répondre au problème des `PTR` automatiques pour le mode *stateless* :

14. http://0pointer.de/lennart/projects/nss-mdns/

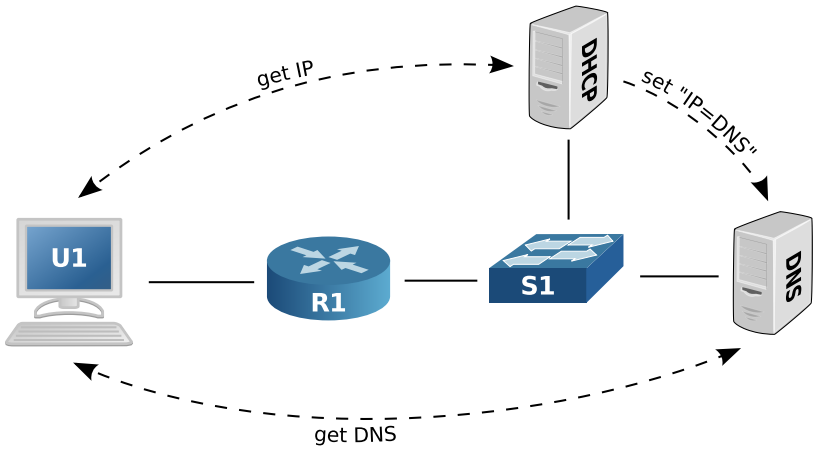


Figure 4.11 – *Injections dynamiques d'enregistrements DNS par le DHCP.*

- **AllKnowingDNS**[15] **:** Propose de répondre à la volée des PTR basées sur les adresses IPv6 demandées, en lui déléguant la zone *.ip6.arpa*.
- **gen6dns**[16] **:** Le fonctionnement est plus simple, puisqu'il s'agit d'un utilitaire destiné à générer directement la base de données du serveur DNS, en créant les PTR de toutes les adresses possibles pour un préfixe donné.
- **RFC 4472 :** Commentaires informationnels concernant les problèmes liés aux DNS dans l'utilisation des IPv6 en production.

15. http://all-knowing-dns.zekjur.net
16. http://www.hznet.de/tools.html

5

IPv6 dans un monde IPv4

« *We need to be the change we wish to see in the world.* » [1] - Gandhi

5.1 Cohabitation (double pile)

Il s'agit de la solution la plus massivement utilisée pour la configuration des machines : le système gère une double pile (*dual-stack*), en autorisant les communications IPv4 et IPv6. Tous les systèmes susévoqués qui supportent l'IPv6 implémentent la double pile, qui est active par défaut à partir de l'instant où au moins une adresse de chaque est renseignée.

Si l'interface doit contacter un hôte en utilisant une IPv4, elle utilisera son IPv4. Si c'est une IPv6, elle choisira une IPv6 pour sortir. Dans le cas où un nom de domaine est utilisé, si celui-ci ne possède qu'un champ `A` elle utilisera son IPv4 et si elle ne possède qu'un `AAAA` ce sera une IPv6.

Les critères de sélection pour l'IPv6 de sortie sont rappelés dans la section 2.8 page 23. Ces mêmes critères permettent de définir le choix entre l'IPv4 et l'IPv6 en cas de double réponse du DNS. Certaines applications comme le navigateur web Safari contournent ces critères en sélectionnant une IP par version puis en testant celle qui répond le plus rapidement.

Cette solution est simple à mettre en œuvre et convient parfaitement aux particuliers en permettant de faire cohabiter des applications inaptes à l'IPv6 avec d'autres plus modernes. Dans le cas de l'administration d'un parc informatique, cette cohabitation impose de configurer deux fois chaque machine, d'avoir deux plans d'adressage fonctionnels et de contrôler deux jeux de règles pour le pare-feu. Au niveau des machines, la double pile entraine tacitement une charge supplémentaire, qui sera particulièrement lourde lorsqu'une application tentera une connexion en IPv6 qui échouera au bout d'un certain temps, avant de retenter en IPv4 (la règle sur les systèmes GNU/Linux est en général de toujours préférer la version IPv6).

1. Dans *Arun Gandhi Shares the Mahatma's Message*, Michel W. Potts, India.

La numérotation du parc entièrement en IPv6 est tentante, mais ne permettra pas aux utilisateurs de traverser les océans IPv4 pour atteindre les îles IPv6 (jusqu'au jour où le problème sera inversé), ni de contacter directement une machine qui ne dispose pas d'une adresse digne de notre nouveau millénaire.

La première partie des solutions présentées ci-dessous expose des solutions pour utiliser de l'IPv6 lorsque les utilisateurs n'ont qu'un accès IPv4 (avec une double pile pour pouvoir utiliser celui-ci). La seconde partie s'intéresse aux solutions possibles pour passer l'ensemble de son parc en IPv6 sans double pile, tout en permettant les connexions vers des serveurs IPv4.

5.2 Faire de l'IPv6 sur un réseau IPv4 (protocole 41)

5.2.1 Encapsulation IP

Cette section traite des tunnels avec encapsulation IP utilisant le protocole 41, destiné à faire passer des paquets IPv6 au-dessus de réseaux entièrement IPv4 (typiquement l'accès à Internet). Les postes clients doivent être au minimum capables de faire de l'IPv6, et devront disposer d'une double pile s'ils veulent pouvoir faire de l'IPv4 en même temps.

Au niveau des entêtes IP, le protocole 16 correspond à TCP et le protocole 17 à UDP. L'introduction d'un nouveau protocole numéroté 41, permet de définir un standard (RFC 2473) qui modifiera le comportement des routeurs, selon le type de tunnel utilisé.

Le principe de base est le même pour tous : deux réseaux entièrement IPv6 sont reliés par un réseau IPv4. Le routeur de sortie (R1) du premier réseau encapsule les paquets IPv6 dans des paquets IPv4 (c'est à dire qu'il ajoute les entêtes IPv4 au début du paquet IPv6, qui se retrouve alors dans sa zone de données/payload), le paquet transite sans problème au travers des réseaux IPv4, pour enfin être désencapsulé par le routeur d'entrée (R2) du second réseau IPv6 (suppression des entêtes IPv4) et transmis tel quel. Un exemple simple est illustré en figure 5.1.

Les extrémités du tunnel peuvent aussi être prises en charge par les pare-feux si ceux-ci le permettent. L'encapsulation (mise en tunnel) d'un paquet comptant pour un franchissement de routeur, le *Hop Limit* est incrémenté de 1. Puisque les paquets ainsi encapsulés ajoutent des octets (entêtes IPv4) aux paquets, il faudra parfois fragmenter pour respecter la MTU entre les deux routeurs (ou l'augmenter à 1280 quand c'est possible, au détriment des performances).

Il existe différentes façons d'utiliser ce mécanisme, qui sont détaillées dans les sections suivantes.

5.2.2 Tunnels statiques

Il est possible de créer des tunnels statiques respectant *stricto sensu* le schéma ci-avant, en configurant les routeurs à la main.

Les machines U1 et U2 ne connaissent que l'IPv6, le tunnel est totalement transparent pour eux et ne nécessite aucune configuration à leur niveau. Par contre, si des réécritures interviennent durant le trajet,

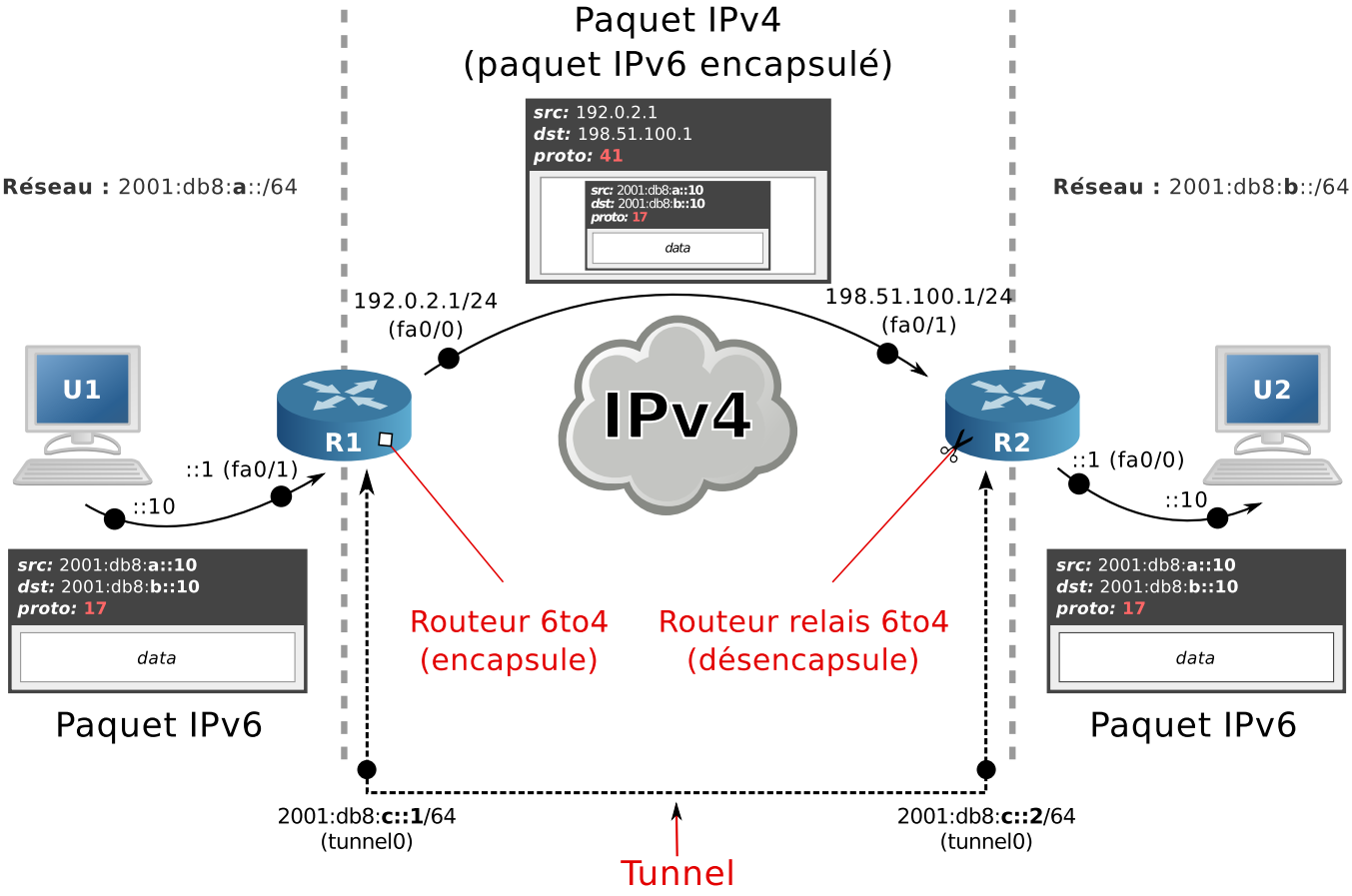


Figure 5.1 – *Fonctionnement de base d'un tunnel IPv6 pour l'IPv4.*

le transfert peut être difficile, notamment à cause des NAT [2].

Les tunnels statiques de ce type correspondent au cas où il faut relier deux sites IPv6 séparés par un réseau IPv4. Mais il faudra monter ce type de tunnel manuellement pour chacune des destinations IPv6 à joindre, ce qui est impossible dans le cas du surf web imprédictible d'un des deux utilisateurs. Pour automatiser la création de ces tunnels, différentes solutions décrites dans la suite sont envisageables.

Une expérimentation mettant en place un tunnel statique est proposée en section 8.3 page 91.

5.2.3 Tunnels 6to4

Afin de permettre la création de ces tunnels à la volée, la RFC 3056 propose d'utiliser un préfixe standardisé par l'IANA : 2002::/16.

Comme dans l'exemple précédent, les deux routeurs devront posséder une adresse IPv4 publique. C'est à partir de celle-ci que sera déduit tout le sous-réseau (annoncé par le routeur avec des *Router Advertisements* ou du DHCPv6) qui se trouvera derrière. Pour se faire, il suffit d'ajouter la valeur en hexadécimale de cette IP à la suite du préfixe donné par l'IANA. Le résultat sera un préfixe sur 48 bits.

2. Un document de l'IETF de 2003 traite de ce problème en détail : http://tools.ietf.org/html/draft-palet-v6ops-proto41-nat-03.

L'outil `ipv6calc` déjà présenté permet de faire ça facilement (exemple avec l'adresse de R1) :

```
$ ipv6calc --in ipv4addr --out ipv6addr --action conv6to4 193.100.0.1
2002:c164:1::
```

L'adresse IPv6 de la machine U1 qui se trouve derrière R1 pourrait donc avoir comme adresse : `2002:c164:1::10/48`.

Une autre solution consiste à laisser l'IPv4 sous forme décimale dans l'IPv6 pour laisser le système calculer lui-même :

```
$ ping6 2001:db8:b::193.100.0.1
PING 2001:db8:b::193.100.0.1(2001:db8:b::c164:1) 56 data bytes
```

Dans le cas de la migration d'un parc informatique, plusieurs habitudes se retrouvent souvent :

1. Recopier l'intégralité de l'ancienne IPv4 à la fin de la nouvelle IPv6 : `192.168.1.10` $\mapsto$ `::c0a8:10a` qui peut aussi être notée `::192.168.1.10`, pour les nostalgiques.
2. Reporter le dernier octet de l'ancienne adresse IPv4 à la fin de la nouvelle adresse IPv6 : `.10` $\mapsto$ `::10`, ce qui permet de commencer à réellement se détacher de l'ancien modèle IP.
3. Laisser l'autoconfiguration faire son travail, et utiliser les adresses MAC (éventuellement masquées automatiquement), ce qui semble la façon la plus saine de faire dans un environnement IPv6 (conformément aux bonnes pratiques présentées en section 2.9 page 24).

Grâce au préfixe standardisé et l'intégration de l'adresse publique IPv4 du routeur dans toutes les adresses au travers du préfixe du réseau généré, deux routeurs qui supportent le 6to4 communiquent parfaitement entre eux au travers d'un réseau IPv4 (on admet que R1 et R2 sont dans le schéma utilisé jusqu'à lors sont des routeurs 6to4) :

1. U1 envoie un paquet IPv6 à destination d'une autre adresse en `2002:`, qu'il confie donc à sa passerelle par défaut R1.
2. Puisque l'adresse de destination n'est pas locale, R1 se prépare à envoyer le paquet vers le réseau mondial.
3. Puisque l'adresse de destination est en `2002:`, il en déduit qu'il faut utiliser le mécanisme 6to4 pour envoyer ce paquet sur sa patte IPv4 reliée à Internet.
4. Il commence par extraire l'adresse IPv4 du routeur 6to4 de destination, grâce aux 4 octets suivant le préfixe de l'IANA.
5. Il encapsule le paquet IPv6 dans un paquet IPv4 en utilisant le protocole 41 présenté ci-dessus, et l'adresse à l'IPv4 extraite correspondant à R2.
6. R2 reçoit ce paquet, constate qu'il s'agit du protocole 41 et désencapsule donc le paquet IPv6.
7. Par mesure de sécurité, il vérifie si les 4 octets après `2002:` de l'IPv6 source correspondent bien à l'adresse IPv4 source du paquet avant désencapsulation.
8. Si c'est le cas, il transmet le paquet IPv6 tel quel à la machine de destination appartenant à son réseau.

Un exemple est donné en figure 5.2 page 57.

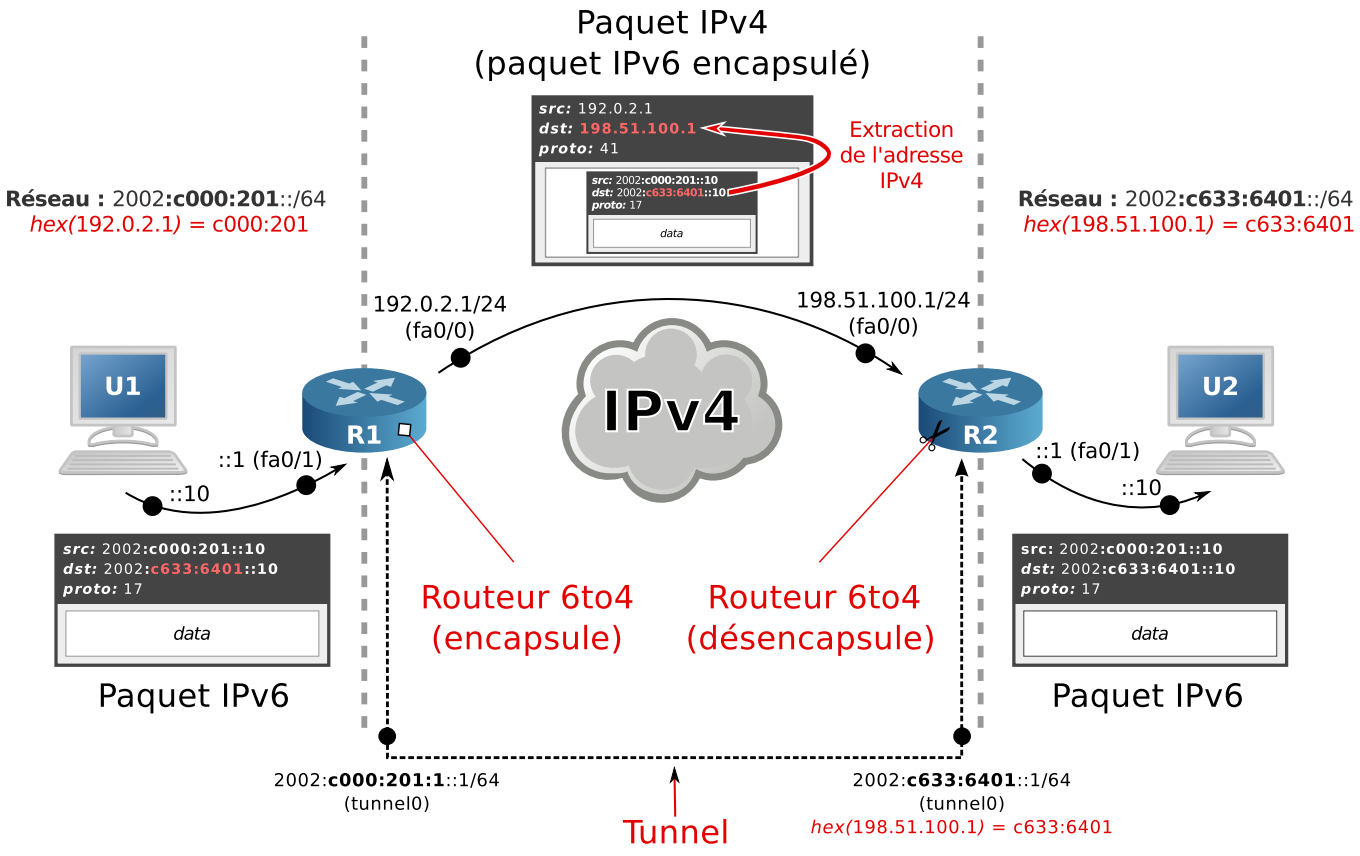


Figure 5.2 – *Communication d'un nœud 6to4 à un autre.*

Ce mécanisme fonctionne parfaitement de routeur 6to4 à routeur 6to4, mais ne permet pas de contacter une adresse IPv6 qui n'est pas en 2002: et qui ne dispose donc pas d'un tel routeur. Il faudra alors utiliser des relais 6to4.

Comme illustré en figure 5.3 page 58, ces relais fonctionneront comme de simples passerelles par défaut à utiliser dans le cas d'une adresse IPv6 qui n'est pas dans le réseau local et qui n'utilise pas le préfixe 6to4.

Le relais à utiliser peut être défini statiquement au niveau du routeur, ou pointer sur l'adresse anycast `192.88.99.1` (transformée en son équivalent 6to4 `2002:c058:6301::`) de la RFC 3068 qui chargera le réseau de trouver le relais le plus proche.

Un relais 6to4 n'annonce que la route `2002::/16` sans la redécouper du côté de l'Internet IPv6, afin de ne pas polluer les tables de routage IPv6 avec des adresses issues d'adresse IPv4. Puisque le retour nécessite aussi un relais qui sera choisi d'une façon imprédictible, la route peut être asymétrique et entraîner des problèmes (notamment pour la traçabilité). De plus, le relais externe peut poser de gros soucis de sécurité (vie privée comme déni de service).

Si U1 (et U2 dans les exemples précédents) souhaite contacter une machine IPv4, il devra disposer en plus d'une double pile avec un double adressage de la part du routeur.

Une expérimentation du 6to4 avec un relais est proposée en section 8.4 page 93.

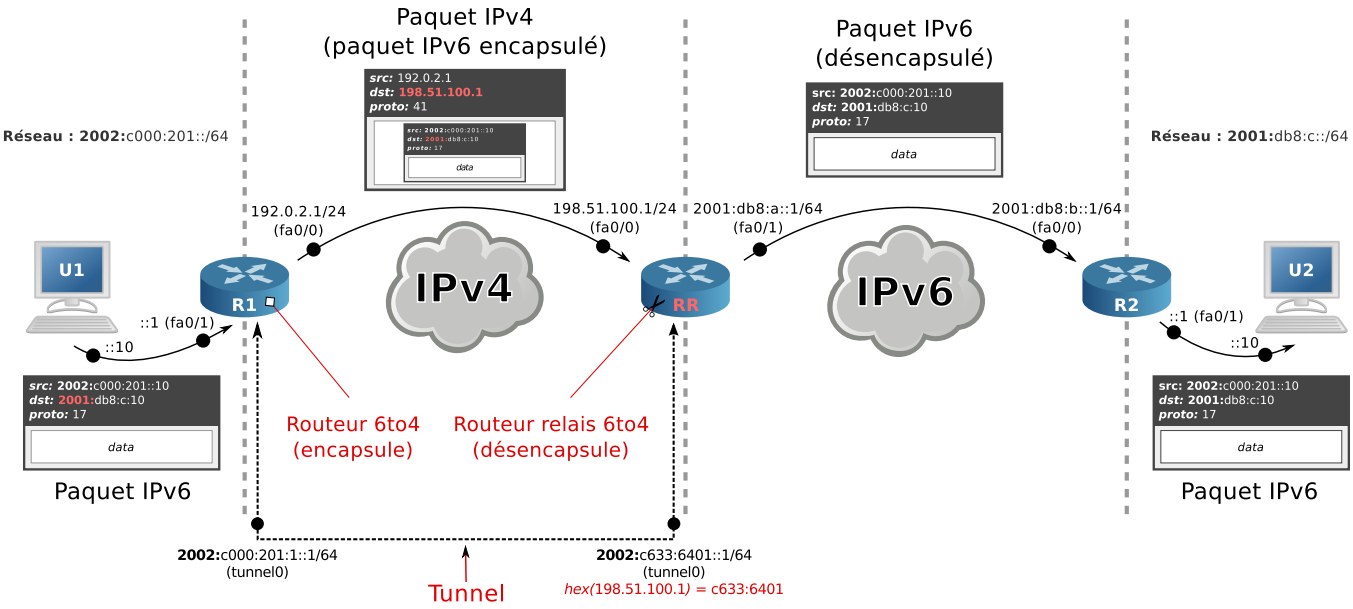


Figure 5.3 – *Communication d'un nœud 6to4 à un nœud IPv6 natif à l'aide d'un relais.*

5.2.4 Tunnels 6rd

Le principal problème du 6to4 réside dans l'impossibilité de contrôler les relais qui mènent à l'Internet IPv6, entraînant des problèmes de stabilité et de sécurité. Le protocole 6rd a été conçu dans le but de pallier ce problème, en permettant au FAI de gérer ses propres relais dans ses propres domaines.

L'innovation a été conduite par le français Rémi Després, et testé pour la première fois par le FAI français Free pour proposer des adresses IPv6 à ses abonnés. En cinq semaines, depuis la proposition de Rémi Després à Free jusqu'à l'annonce officielle aux abonnés en passant par toutes les étapes de décisions hiérarchiques et de marketing, le second plus gros fournisseur d'accès à haut débit français a pu proposer une connectivité IPv6 totalement transparente pour l'ensemble de ses abonnés. Le succès d'une telle opération a eu un retentissement mondial et a valu à l'IETF de créer la RFC 5969 qui normalise le protocole en évoquant « *a successful commercial "rapid deployment" of the 6rd mechanism by a residential service provider* ».

Le 6rd est une application du 6to4, à ces différences près :

- Les préfixes IPv6 utilisés sont issus des préfixes qui appartiennent au FAI, plutôt que des sous-ensembles du préfixe normalisé 2002::/16 du 6to4.
- Il est donc impossible de différencier du trafic sorti d'un réseau 6rd d'un trafic IPv6 natif (ce qui vaut à la France d'avoir artificiellement 95%[3] de son trafic IPv6 soit-disant natif, principalement en provenance de Free).
- Il peut y avoir plusieurs préfixes IPv6 différents, chacun correspondant à un domaine 6rd.
- La taille du préfixe IPv6 est libre.
- La taille allouée à l'adresse IPv4 incorporée est libre aussi : si l'ensemble des adresses IPv4 du domaine peuvent être agrégées en un sous réseau, le préfixe de celui-ci n'a pas besoin d'être

3. Page 15 de http://meetings.ripe.net/ripe-57/presentations/Colitti-Global_IPv6_statistics_-_Measuring_the_current_state_of_IPv6_for_ordinary_users_.7gzD.pdf

renseigné (par exemple, si toutes les adresses appartiennent à `10.0.0.0/8`, seuls les 24 derniers bits sont utiles).

Ces différences ont fait le succès du 6rd :

- En utilisant ses propres préfixes, le FAI contrôle tous les relais, ce qui annihile la plupart des défauts du 6to4.
- Les paquets encapsulés étiquetés protocole 41 ne sortent donc pas de l'infrastructure du FAI.
- Ainsi la MTU des paquets IPv4 peut être augmentée (ou celle des IPv6 diminuée) pour empêcher la fragmentation liée aux doubles entêtes.
- Les relais peuvent être joignables avec des adresses anycast propres au FAI (une par domaine 6rd), offrant ainsi des possibilités de redondance automatique.
- L'IANA a ajouté une option *OPTION_6RD* (section 7.1.1 de la RFC 5969) pour configurer automatiquement un routeur pour un domaine 6rd précis.
- Les routeurs des abonnés peuvent facilement ajouter un support 6to4 (avec des adresses supplémentaires qui utilisent le préfixe 6to4) qui sera utilisé pour contacter toutes les machines en `2002::/16` sans passer par les relais.

Contraintes à respecter :

- La taille du préfixe IPv6 plus le nombre de bits nécessaires pour représenter l'IPv4 incluse ne doit pas dépasser 64 bits pour permettre l'autoconfiguration à partir des adresses MAC sur les réseaux des abonnés.
- À cause des adresses anycast utilisées pour les relais, les paquets IPv4 utilisant le protocole 41 ne doivent pas subir de fragmentation (il n'y a pas de garantie que deux relais partageant la même adresse anycast n'utilisent pas le même identifiant de fragmentation, ce qui entraînerait des recompositions hasardeuses).
- Il doit y avoir un domaine 6rd (donc un préfixe IPv6) interne si le FAI utilise du NAT en sortie.

Inconvénients :

- Pour respecter la contrainte des 64 bits maximum, les préfixes IPv6 des domaines 6rd doivent être relativement petits, ce qui implique une consommation assez importante du préfixe total alloué au FAI (qui peut aussi nécessiter des préfixes pour de l'IPv6 natif).
- Comme pour le 6to4, les adresses IPv4 devraient être fixes pour faciliter la traçabilité.

L'abonné devra bénéficier d'une double pile pour pouvoir continuer à accéder à l'Internet IPv4.

Du côté du FAI, cette solution lui permet de proposer une connectivité IPv6 à ses abonnés en un temps record, mais ne lui permet pas d'être prêt pour l'avenir : si un *sniffer* était placé entre le boîtier ADSL de l'abonné et la prise murale, il n'y verrait pas un seul paquet IPv6, puisque tout est encapsulé et que le réseau du FAI est resté entièrement en IPv4. Un exemple de communication avec l'extérieur d'un réseau 6rd est disponible en figure 5.4.

La nouvelle option de DHCPv4 (`option-code`) *OPTION_6rd* permet d'ajouter les champs suivants au annonces DHCP :

`IPv4MaskLen` : Valeur entre 0 et 32 qui représente le nombre de bits de poids fort qui sont commun à toutes les IPv4 du domaine 6rd (donc la représentation des IPv4 dans les IPv6 se fera sur 32 - `IPv4MaskLen` bits).

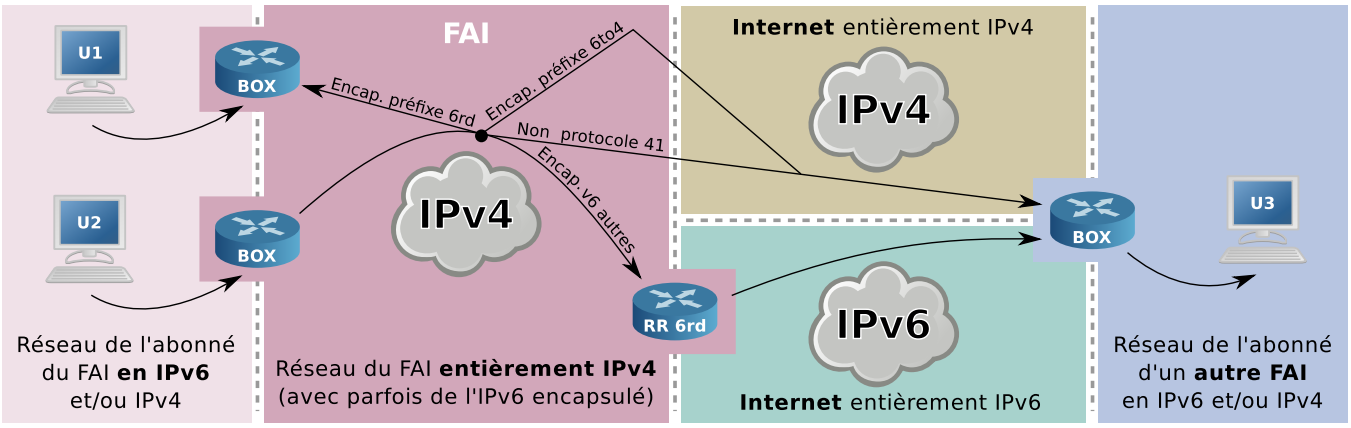


Figure 5.4 – *Communication IPv6 via un réseau 6rd.*

`6rdPrefixLen` : Le préfixe IPv6 dédié au domaine 6rd annoncé (32 - `IPv4MaskLen` + `6rdPrefixLen` doit donc être inférieur ou égal à 128 et idéalement à 64).

`6rdBRIPv4Address` : Une ou plusieurs adresses IPv4 (éventuellement anycast) qui permettent de contacter les relais pour le domaine 6rd.

Un exemple d'expérimentation du 6rd avec un relais anycast est proposé en section 8.5 page 96.

5.2.5 Tunnels ISATAP

Les tunnels ISATAP (*Intra-Site Automatic Tunnel Protcol*) sont destinés à être utilisés à l'intérieur d'un site, pour relier des machines obligatoirement en double pile au travers d'un tunnel IPv4. Ce sont les machines qui établissent le tunnel entre elles et non les équipements réseau. Le tunnel utilise un protocole de couche liaison de données, et permet donc aux machines de se considérer directement sur le même lien. Ce type de tunnel ne passe pas les NAT, il doivent donc être utilisés soit dans un adressage entièrement privé (éventuellement avec un VPN) soit dans un adressage entièrement public.

Chaque machine client doit supporter ISATAP, qui est disponible sous Windows (à partir de XP), GNU/Linux et Cisco mais en version *pre-alpha*[4] pour Mac OS. Pour ce qui est des téléphones IP, webcams, imprimantes et autres, l'utilisation de ce genre de tunnel est impossible.

Le protocole (RFC 5214) n'impose pas de préfixe comme c'est le cas pour le 6to4, mais un format spécifique pour les 8 derniers octets (figure 5.5 page 62).

Les adresses IPv6 contiennent donc les adresses IPv4 des hôtes, ce qui permet de les extraire comme dans le cas du 6to4. L'algorithme d'extraction est illustré en figure 5.6 page 62.

Le protocole de couche 2 utilisé n'est pas ethernet mais un lien local NBMA *NonBroadcast Multiple-Access* (comme pour le 6rd) : le *nonbroadcast* implique l'inopérance du multicast. Puisque le passage des paquets sur le réseau se fait intégralement en IPv4, le NDP n'est pas nécessaire pour la résolution des adresses MAC.

4. http://www.momose.org/macosx/isatap.html

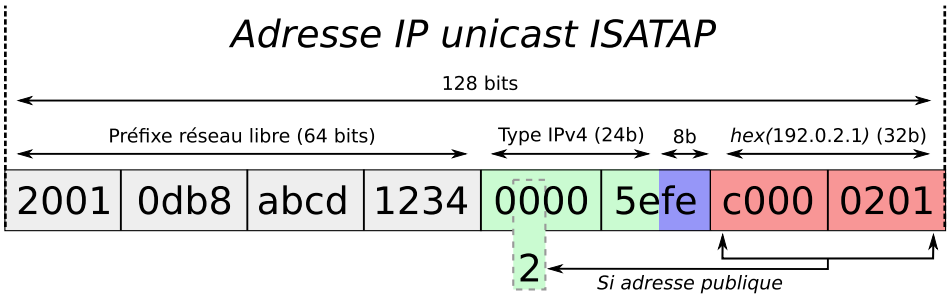


Figure 5.5 – *Confection d'une adresse ISATAP.*

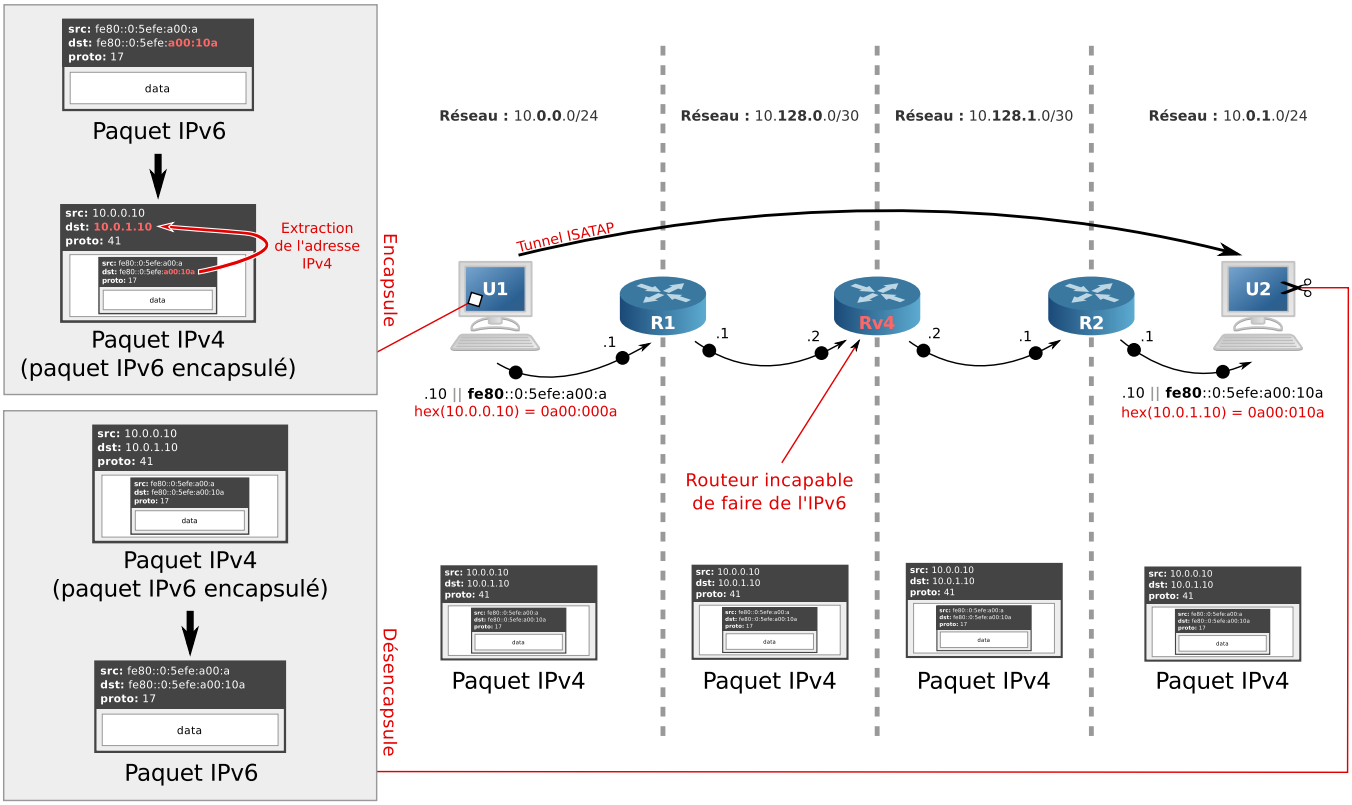


Figure 5.6 – *Fonctionnement de ISATAP.*

La RFC 5214 prévoit également un mécanisme de passerelle ISATAP pour communiquer avec le monde IPv6 extérieur. Un serveur DNS est utilisé pour gérer la découverte des routeurs ISATAP, ce qui ressemble à un viol des concepts fondamentaux de conception des réseaux, avec un étrange mélange des couches.

En fonctionnant ainsi en couche 2 et en n'ayant besoin d'aucune intervention sur les équipements réseaux (qui doivent tout de même autoriser le transport du protocole 41), ISATAP est un très bon moyen de virtualiser une connexion *ad-hoc* en IPv6, entre deux machines séparées par un réseau IPv4 qui supportent le protocole. Passé ce cas d'utilisation, les contraintes imposées et les moyens à déployer pour faire plus, le laissent sans avenir face au 6to4/6rd.

5.3 Autres solutions pour faire de l'IPv6 sur un réseau IPv4

5.3.1 Encapsulation UDP pour mieux passer les NAT (Teredo)

Dans le cas de l'utilisation de NAT pour les réseaux IPv4, les tunnels 6to4 et ses dérivés imposent aux routeurs les gérant d'implémenter directement les fonctionnalités d'encapsulation. D'une manière générale, tous les routeurs ne proposent pas ces options, et peuvent être une barrière à l'établissement des tunnels. L'encapsulation UDP (couche 4) plutôt que IP (couche 3) permet non seulement de passer les NAT facilement, mais aussi les pare-feux qui n'ont plus besoin d'être configurés pour laisser passer le protocole 41.

Teredo est une solution proposée par Microsoft standardisée par l'IETF avec la RFC 4380, qui utilise ce type d'encapsulation et qui est disponible sur la plupart des versions des systèmes d'exploitation qui supportent l'IPv6 (l'avantage d'une solution proposée par Microsoft étant que Windows ne freine pas les autres systèmes).

Son fonctionnement n'est pas très différent des autres types de tunnels :

- Préfixe IPv6 `2001:0000::/32` spécifique à Teredo qui embarque des adresses IPv4.
- Système client/serveur pour communiquer d'une adresse Teredo à l'autre.
- Relais connectés à l'Internet IPv6 à contacter pour sortir.

La première étape pour l'établissement d'un tunnel Teredo est de permettre à la machine de découvrir le type du NAT derrière lequel elle se trouve (`ip1` et `p1` sont *mappés* automatiquement au niveau du NAT sur des équivalents privés, et chaque envoi en UDP donne lieu à l'établissement de règles dynamiques de NAT plus ou moins strictes pour permettre au serveur de répondre). Les types de NAT sont détaillés dans le tableau 5.1.

Type de NAT	Envoi	Conditions de retour	Commentaire
full-cone	`ip1:p1` ↦ `ip2:p2`	`*:*` ↦ `ip1:p1`	Type de NAT des boîtiers ADSL.
symétrique	`ip1:p1` ↦ `ip2:p2`	`ip2:p2` ↦ `ip1:p1`	Une `ip1` publique différente pour chaque connexion.
restricted cone	`ip1:p1` ↦ `ip2:p2`	`ip2:*` ↦ `ip1:p1`	
restricted port	`ip1:p1` ↦ `ip2:p2`	`ip2:p2` ↦ `ip1:p1`	

Table 5.1 – *Les différents types de NAT (échanges UDP).*

Puisque l'UDP donne lieu à la création de règles NAT automatiques, c'est un excellent moyen (appelé *hole punching*) de passer les NAT sans devoir configurer une règle PAT avec un port statique à la main, à condition que des paquets soient régulièrement envoyés pour maintenir le *mappage* actif. C'est la raison pour laquelle les protocoles comme SIP (téléphonie IP) utilisent beaucoup cette technique qui permet de ne pas contraindre les utilisateurs lambda à devoir aller manipuler leur boîtier ADSL.

Pour déterminer le type de NAT derrière lequel l'utilisateur se trouve, les logiciels clients utilisent un mécanisme de type STUN [5] (*Simple Traversal of UDP through NAT*) simplifié (section 5.2.1 de la RFC

5. http://upload.wikimedia.org/wikipedia/commons/6/63/STUN_Algorithm3.svg

4380). Il s'agit d'une série de questions-réponses depuis le NAT vers un serveur STUN (dans notre cas, ce sont les serveurs Teredo qui font office) qui visent à déterminer les restrictions en fonction des paquets reçus ou non. Les réponses STUN permettent aussi de connaître son IP ainsi que son port public.

À partir des informations déduites des échanges STUN, l'utilisateur est à même de construire son adresse IPv6 Teredo, en suivant le schéma disponible en figure 5.7 page 64.

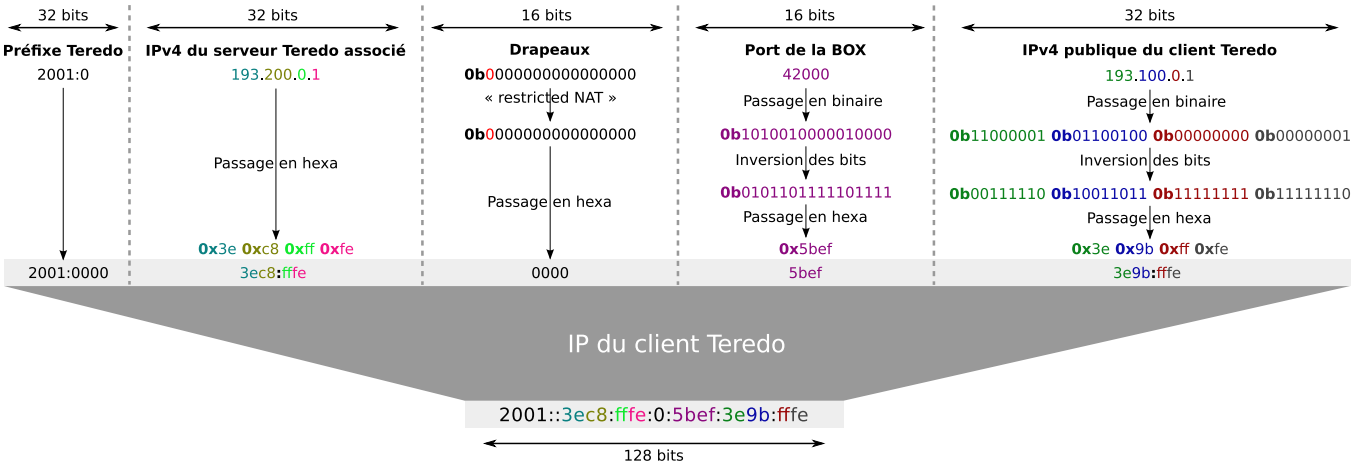


Figure 5.7 – *Confection d'une adresse Teredo.*

Dans le cas d'un NAT *full-cone* (simplement appelé *cone NAT* dans la littérature de Microsoft, le reste étant des `restricted NAT`), le drapeau permet au relais de savoir qu'il n'a pas besoin de passer par des échanges avec le serveur pour transmettre son paquet encapsulé (les échanges réguliers du client avec le serveur permettent de laisser le port UDP *mappé* - et renseigné dans les adresses - dans le NAT, et la nature de celui-ci permet à n'importe qui de l'exploiter).

Si un NAT symétrique est détecté, le client Teredo avertit l'utilisateur qu'il ne pourra pas faire de tunnel Teredo[6].

Enfin, s'il s'agit d'un NAT *restricted*, le protocole prévoit un échange de paquets *bubble* en utilisant le serveur, qui permettront de demander au client d'envoyer un paquet UDP directement au relais. Ainsi, un nouveau *hole punching* sera mis en place et le relais pourra déterminer le port à utiliser pour une communication directe (il devra reconnaitre le client grâce à son adresse IP, ce qui explique pourquoi le NAT symétrique n'est pas supporté).

Les serveurs Teredo ne sont pas destinés à faire transiter du trafic. Contrairement aux relais, ils ne sont destinés qu'à recevoir des paquets de taille minimale : ceux permettant de garder le port UDP *mappé*, ceux permettant de demander un contact client-relais (paquets *bubbles*) et ceux destinés à faire des requêtes de `ping` vers d'autres hôtes.

Un client Teredo commence par trouver son adresse et initier un tunnel avec le serveur (figure 5.8 page 65). Il pourra ensuite discuter via le tunnel réservé, y compris à travers un NAT *restricted* (figure 5.9 page 65).

6. Une extension SymTeredo a été proposée pour résoudre ce cas : http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?reload=true&arnumber=1633339.

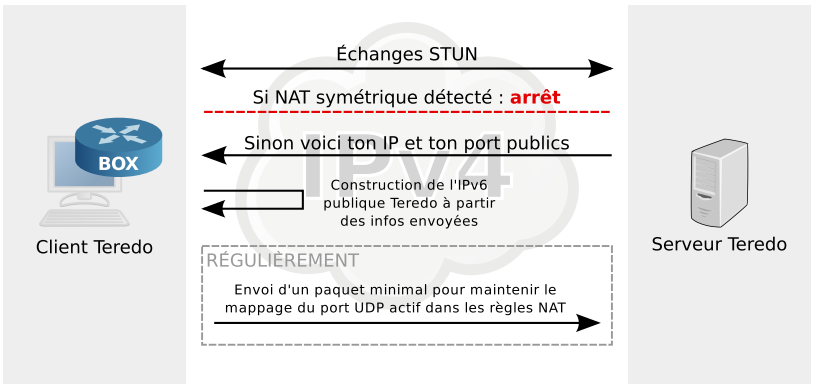


Figure 5.8 – *Découverte de l'adresse Teredo et réservation d'un tunnel.*

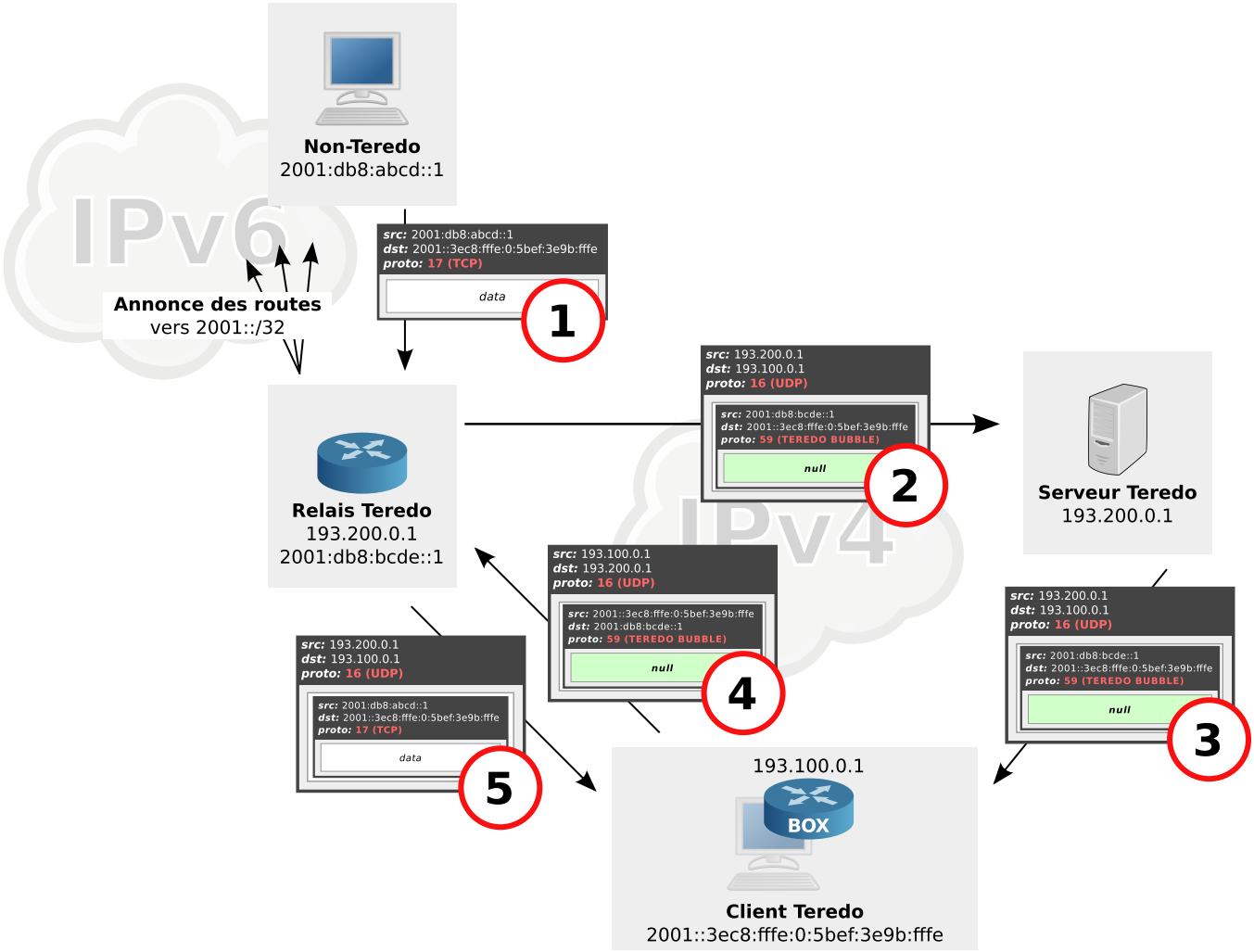


Figure 5.9 – *Fonctionnement de Teredo avec un NAT de type* restricted.

Explications pour la figure 5.9 page 65 :

1. Le client IPv6 natif envoie son paquet IPv6 à destination du client Teredo en l'adressant à un relais qui annonce le préfixe Teredo, comme à n'importe quel routeur.

2. Après une série de vérifications, le relais extrait le drapeau de NAT et conclut qu'il s'agit d'une machine derrière un NAT *restricted* (bit à 0). Ne pouvant pas lui transmettre directement le paquet, il extrait l'adresse de son serveur Teredo associé, et lui envoi un paquet *bubble* encapsulé dans un paquet UDP IPv4.
3. Le serveur Teredo se contente de router le paquet, ayant l'autorisation d'utiliser le port UDP *mappé*.
4. Le client désencapsule le paquet UDP et trouve un paquet IPv6 *bubble* : il envoie donc à son tour un paquet encapsulé de même nature au relais Teredo (dont il détermine l'adresse à partir des IP sources du premier paquet), créant ainsi une règle dynamique de retour dans le NAT, avec un nouveau port.
5. Le relais associe la provenance du paquet *bubble* avec celui qu'il vient d'envoyer grâce à l'IP du client, et détermine le port à utiliser grâce au port source du paquet qu'il vient de recevoir. Il peut alors directement transmettre le paquet de l'hôte natif IPv6, via le nouveau port. Il stockera ces informations dans une table avec un temps d'expiration, pour éviter de redemander un port à chaque échange.

Les tunnels Teredo sont soumis à de gros problèmes de sécurité [7].

5.3.2 Tunnels brokers

Les *tunnels brokers* (RFC 3053) proposent une approche différente, en utilisant un tunnel IPv4 permanent, à l'instar d'un VPN. Une fois le client connecté au *tunnel server*, les échanges en IPv6 se font au travers de la nouvelle interface virtuelle, qui utilise le mode d'encapsulation qu'elle désire.

Pour enregistrer les nouveaux utilisateurs et leur donner accès aux *tunnels servers*, des nœuds connus de type *broker* sont utilisés. Ceux-ci seront éventuellement payants, et se chargeront en plus d'attribuer les adresses IPv6 (tirées d'un préfixe lui appartenant, formatées comme bon lui semble) et d'indiquer au client comment établir le tunnel avec le *tunnel server* indiqué. Les clients doivent donc posséder le logiciel spécifique au *tunnel broker*, qui lui permettra d'établir la liaison avec le *tunnel server*. Un exemple est donné en figure 5.10 page 67.

Cette solution très sécurisée (les tunnels peuvent être aussi chiffrés) nécessite plus de ressources que les autres, et n'est destinée qu'aux particuliers isolés.

Quelques *brokers* connus :

- SixXS : `http://www.sixxs.net`.
- Freenet6 : `http://go6.net`.
- Hurricane Electric : `http://tunnelbroker.net`.
- BT IPv6 : `http://www.ipv6.bt.com`.

Plusieurs *brokers* disposent d'un logiciel client compatible directement dans (par exemple) les dépôts Debian :

- Paquet *aiccu* pour SixXS.

7. `http://www.symantec.com/avcenter/reference/Teredo_Security.pdf`

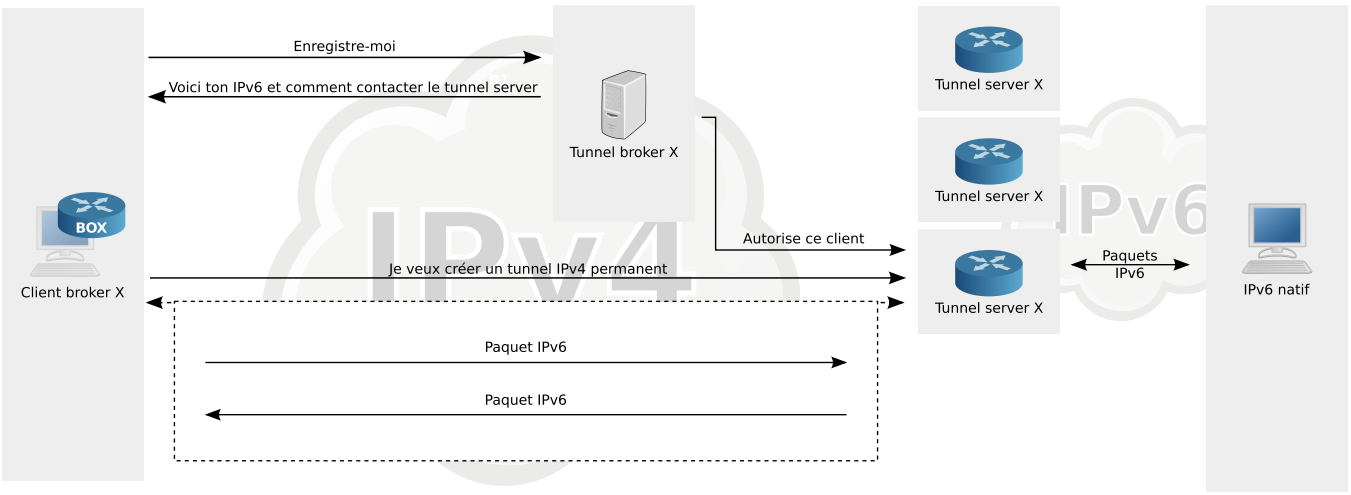


Figure 5.10 – *Fonctionnement d'un* tunnel broker.

– Paquet *gogoc* (protocole TSP) pour Freenet.

Si cette solution sécurise les échanges, elle n'est pour autant pas garante de la protection de la vie privée ou de la neutralité du Net, si l'utilisateur utilise un *broker* qu'il ne contrôle pas.

5.3.3 Sans encapsulation avec le NAT64/DNS64

Ces technologies concernent en général plutôt les réseaux entièrement IPv6, qui ont une connexion IPv6 à disposition, mais qui souhaitent continuer à pouvoir contacter des IPv4. Toutefois, lorsque le DNS64 est configuré pour transformer l'intégralité des `A` en faux `AAAA`, le réseau n'a plus besoin de la connectivité IPv4.

La différence fondamentale avec la plupart des technologies présentées ici, est qu'il n'utilise pas d'encapsulation et ne nécessite que l'utilisation d'un routeur capable de faire du NAT64. Pour plus d'informations sur ces technologies et l'astuce permettant de se passer de connexion IPv6, se reporter à la section 5.4.3 page 68.

5.3.4 Serveurs mandataires (*proxies*)

Passer par une passerelle applicative permet de recevoir des données en IPv4 pour les transmettre en IPv6 au destinataire final, ou inversement. Cette solution est très simple à mettre en œuvre, mais n'est disponible que pour un nombre réduit de protocoles habitués à être proxifiés (HTTP, SMTP, POP, IMAP, FTP, DNS, SIP, etc.).

Les serveurs mandataires ont aussi la fâcheuse tendance à rapidement devenir des goulots d'étranglement.

5.4 Faire de l'IPv4 sur un réseau IPv6

5.4.1 Généralités

Toutes les solutions abordées ci-avant nécessitent une double pile pour faire cohabiter l'IPv4 et l'IPv6.

Cette section présente les solutions pour bénéficier d'un réseau entièrement IPv6 (comme dans un 6to4 sans IPv4), tout en laissant la possibilité aux usagers de contacter des machines restées en IPv4 (sans double pile). Cette façon de faire permet de réellement passer son réseau en IPv6, sans doubler sa charge de travail en conservant un double adressage avec un double jeu de règles pour le pare-feu, et sans encapsulation. Il s'agit donc aussi de la meilleure façon de se préparer pour l'avenir.

Ces solutions se basent sur un mécanisme de translation d'adresses.

5.4.2 NAT-PT et NATPT-PT

Ces deux technologies (RFC 2766), souvent reprises dans la littérature, ont été rendues obsolètes par la RFC 4966 et ne doivent plus être employées.

Elles ne seront pas conséquent plus supportées sur les futurs IOS [8] de Cisco.

Il est remplacé avantageusement [9] par le NAT64.

5.4.3 NAT64 / DNS64

Le NAT64 est l'une des dernières solutions disponible pour traduire des paquets IPv6 en IPv4. Malheureusement, ses spécifications n'ont pas encore été officialisées par l'IETF (RFC *PROPOSED STANDARD*). Malgré le retard de la normalisation, des solutions logicielles fonctionnelles sont déjà disponibles.

Le NAT64 utilise le format d'adresses de la RFC 6052 illustré en figure 5.11 page 68, qui décrit comment transformer une IPv4 en IPv6 en utilisant le préfixe `64:ff9b::/96`.

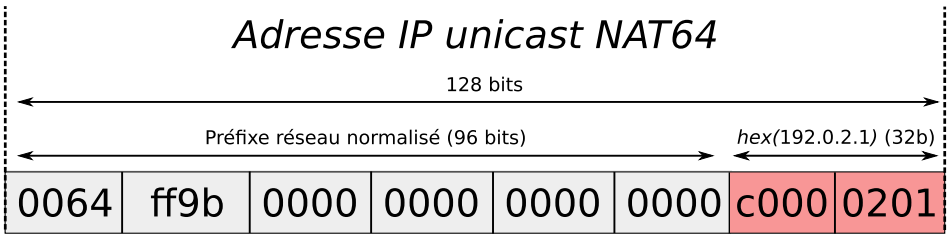


Figure 5.11 – *Confection normalisée d'une adresse IPv6 depuis une adresse IPv4.*

8. Cf. « *Is NAT-PT still supported in IOS (even though it's deprecated by the IETF) ?* » ici : http://www.cisco.com/web/learning/le21/le39/docs/tdw130_qa.pdf
9. http://blog.ioshints.info/2011/03/nat-pt-is-dead-long-live-nat-64.html

Ce préfixe étant destiné à être utilisé uniquement à l'intérieur du réseau, il peut être changé par un préfixe libre, à condition d'ajouter un octet *null* à partir du bit 64 si le préfixe IPv6 est plus petit que 96 (section 2.3 de la RFC 6052).

Le NAT64 va de paire avec le DNS64, illustré en figure 5.12 page 69. Celui-ci est chargé de résoudre les noms de domaine comme n'importe quel serveur DNS. Sauf que si la réponse est en IPv4 uniquement, il transformera cette dernière en IPv6 en respectant un formatage précis. Son ancêtre le NATPT-PT modifiait les réponses DNS provenant de n'importe quel serveur directement dans les paquets, pour modifier les `A` en `AAAA`, ce qui rendait la solution un peu plus transparente, mais bien plus lourde que le DNS64.

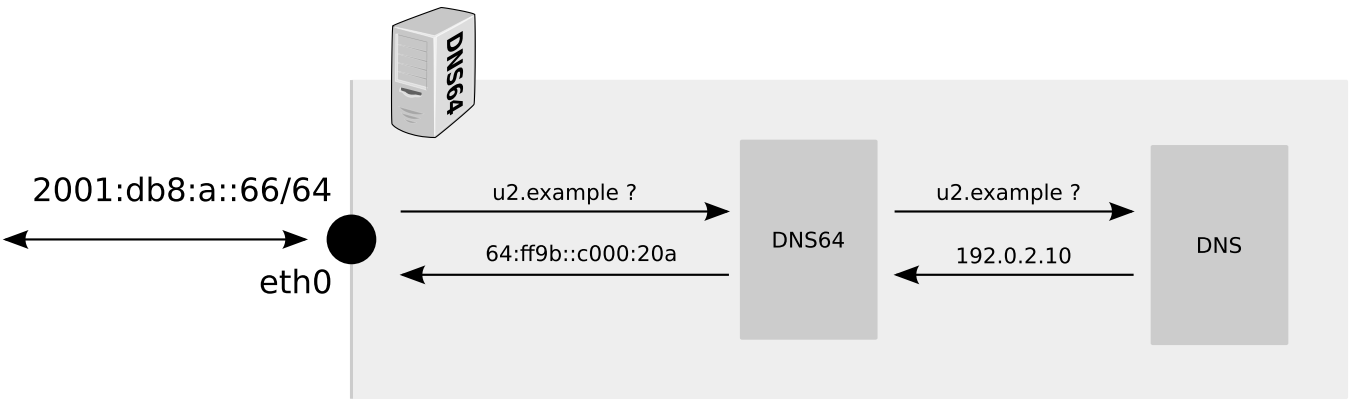


Figure 5.12 – *Fonctionnement d'un DNS64, allié indispensable du NAT64.*

La passerelle du réseau IPv6 doit avoir accès un accès à l'Internet IPv4 en plus de l'IPv6, comme illustré en figure 5.13 page 69.

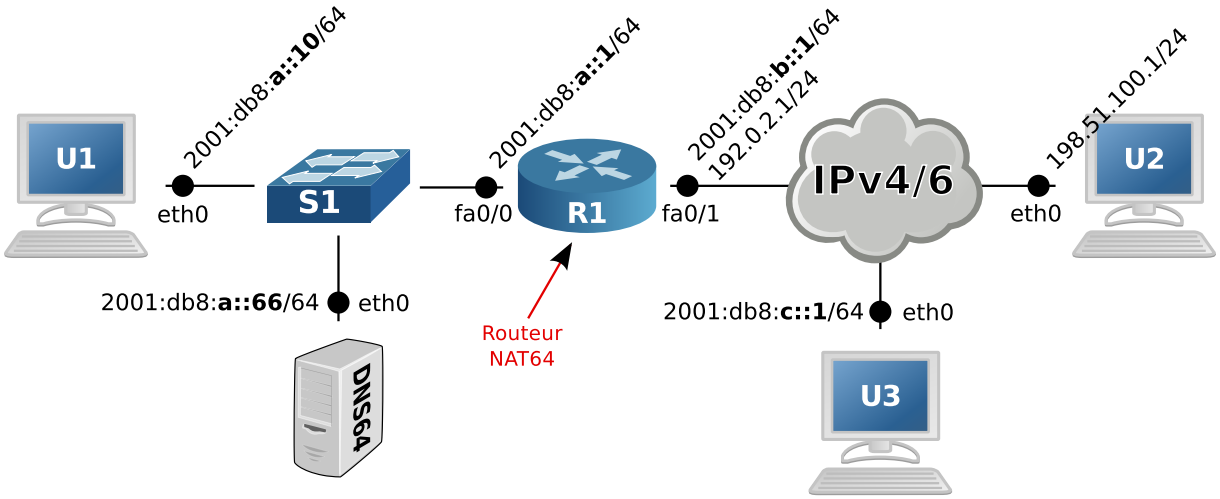


Figure 5.13 – *Fonctionnement du NAT64.*

Lorsqu'un paquet est envoyé à destination d'une adresse IPv6 fictive, la passerelle NAT64 se charge de transformer ce paquet en un paquet IPv4 (sans l'encapsuler) en utilisant l'adresse IPv4 de destination extraite de l'adresse fictive. Elle utilisera une règle de NAT, comme en NAT44, pour être capable d'associer les réponses.

Le NAT64 ***stateless*** utilise une plage IPv4 privée interne pour créer ses règles de NAT (ce qui double les conversions, mais permet de ne pas avoir à stocker l'état des connexions), illustré en figure 5.14 page 70.

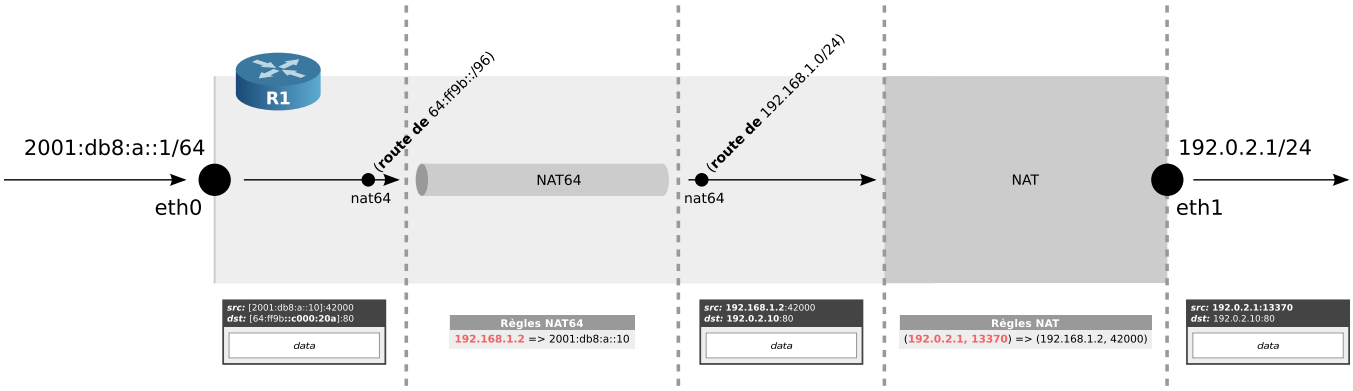


Figure 5.14 – *Mécanismes internes du NAT64* stateless.

Dans cet exemple, le NAT64 *stateless* a choisi l'IP `192.168.1.2` parmi les adresses qu'il avait à disposition et l'a associée de façon unique à l'IPv6.

La version ***stateful*** (RFC 6146 et figure 5.15 page 70) associe directement les IPv6 aux IPv4, en conservant l'état des connexions, comme les NAT/PAT44 habituels.

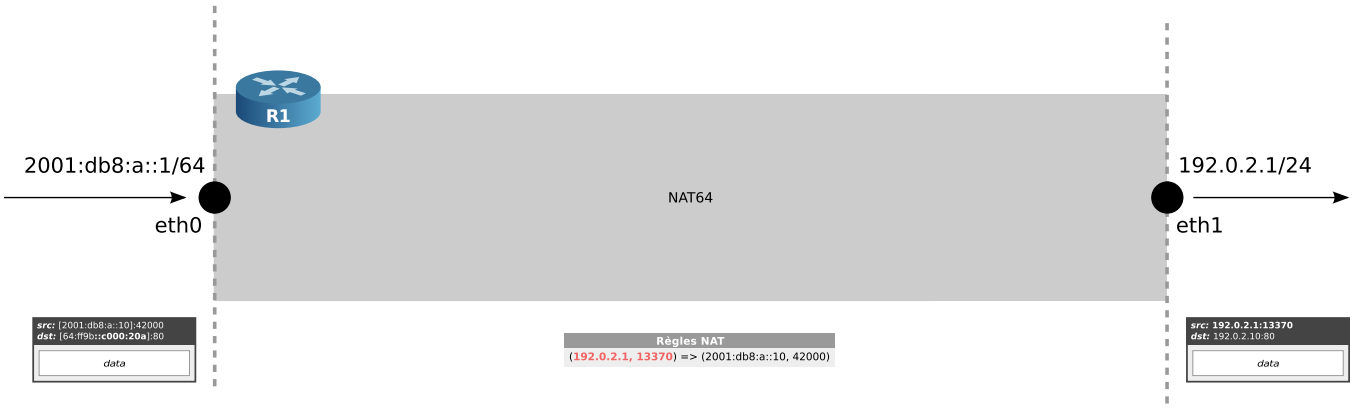


Figure 5.15 – *Mécanismes internes du NAT64* stateful.

Un document intitulé *Performance of NAT64 versus NAT44 in the Context of IPv6 Migration*[10] propose des tests de performance, notamment entre du NAT64 *stateless* et du NAT44 classique. Il utilise les mêmes outils que ceux utilisés dans la partie expérimentations et permet de constater que cette solution pourrait passer l'échelle aussi bien que du NAT classique. À noter tout de même qu'un NAT44 est ajouté au NAT64 pour la version *stateless*.

Côté DNS64, le serveur *Bind* supporte cette fonctionnalité depuis sa version 9.8[11], directement dans ses options (sinon un autre utilitaire peut être employé, les deux cas étant envisagés dans les expérimentations) :

10. http://www.iaeng.org/publication/IMECS2012/IMECS2012_pp638-645.pdf
11. Un chapitre est consacré au DNS64 dans l'Oreilly « *DNS & BIND on IPv6* » , disponible ici : http://my.safaribooksonline.com/book/networking/dns/9781449308025/dns64/27.

```
1       dns64 64:ff9b::/96 {
2    suffix ::;
3               recursive-only yes;
4               clients { ::1/128; };
5               mapped { !10/8; !172.16/12; !192.168/16; any; };
6               exclude { 64:ff9b::/96; };
7    break-dnssec yes;
8       };
```

Toutes ces contraintes sont optionnelles :

suffix : Si le préfixe du DNS64 est inférieur à 96, un suffixe peut être utilisé, sinon la contrainte `::` est positionnée par défaut.

recursive-only : Par défaut à `no`, cette contrainte permet d'indiquer si les transformations s'appliquent aussi sur les zones du Bind en cours, ou non.

clients : Permet de n'activer la fonctionnalité DNS64 que pour un certain nombre de clients DNS (dans cet exemple il s'agit du cas où le Bind est installé en local sur la machine utilisateur, et la valeur par défaut est `any`).

mapped : Un `AAAA` modifié ne sera renvoyé que lorsque ces IP seront découvertes dans la version `A`. Le point d'exclamation devant les trois plages de l'exemple (correspondant aux IP privées de la RFC 1918) permettent d'exclure des IP avant de toutes les accepter.

exclude : Lorsqu'un `AAAA` est découvert, il est retourné sans regarder la version `A`. Sauf pour les adresses comprises dans les plages de cette contrainte, pour lesquelles ont renverra toujours un `AAAA` issu du `A` (dans l'exemple, on décide de recalculer systématiquement soi même les `AAAA` lorsqu'il s'agit manifestement déjà d'un `AAAA` modifié). C'est cette contrainte, lorsqu'elle a la valeur `::/0`, que le NAT64 peut être utilisé avec une connexion sans accès IPv6.

break-dnssec : Lorsque DNSsec est utilisé, les conversions de `A` en `AAAA` ne s'opèrent plus, puisque le DNS64 qui transmets (*forwarder*) va systématiquement casser les vérifications. Ce comportement peut-être inversé avec cette contrainte.

Si les enregistrements `A` ont un champ `PTR`, les `AAAA` modifiés en auront aussi un automatiquement. Le serveur DNS64 répondra toujours avec un `PTR` correspondant à l'IPv6 modifiée, configuré en CNAME du `PTR` de l'IPv4.

Le NAT64/DNS64 a été expérimenté en mode *stateless* en section 8.6.1 page 98 et en mode *stateful* à la section 8.6.2 page 104.

5.5 Récapitulatif

Le tableau 5.2 page 72 donne un aperçu de la mise en concurrence des solutions qui sont à disposition pour faire de l'IPv6 avec ou sans lien IPv6, et en continuant de pouvoir accéder aux services IPv4.

Solution	Dbl pile	Lien IPv6	Encap.	Config.	Transparent	IP natives	Univ.
6to4	Oui	Oui (relais)	Oui (41)	Réseau	Oui	Non	Oui
6rd	Oui	Oui	Oui (41)	Réseau	Oui	Oui	Oui
NAT64	Non	Oui/Non	Non	Réseau	Oui	Oui	Oui
ISATAP	Oui	Non	Oui (41)	Utilisateur	Non	Oui	Oui
Teredo	Oui	Oui (relais)	Oui (UDP)	Utilisateur	Oui (+NAT)	Non	Oui
Brokers	Oui	Non	Oui (TCP)	Utilisateur	Non	Oui	Oui
Proxies	Non	Oui	Non	Utilisateur	Oui	Oui	Non

Table 5.2 – *Comparatif des solutions pour faire cohabiter l'IPv4 et l'IPv6.*

Précisions pour le tableau 5.2 page 72 :

- *Double pile* : Oui s'il faut garder la pile IPv4 pour pouvoir communiquer avec des services IPv4.
- *Lien IPv6* : Oui s'il faut que le réseau ait accès à un lien IPv6 à au moins un endroit, pour que la solution soit mise en place. Lorsque *(relais)* est indiqué, le lien n'est pas nécessaire si l'administrateur décide d'utiliser des relais externes. Le NAT64 peut fonctionner de paire avec un lien IPv6, ou s'en passer totalement.
- *Encapsulation* : Le type d'encapsulation est indiqué entre parenthèses, s'il y a lieu.
- *Configuration* : *Réseau* signifie que la configuration se fait au niveau du routeur, que la solution est accessible à l'ensemble du réseau et que les utilisateurs n'ont pas obligatoirement connaissance de la solution déployée. *Utilisateur* signifie que la machine de l'utilisateur doit être configurée (et compatible) spécifiquement pour cette solution, et qu'elles devront l'être une à une.
- *Transparent* : Oui si les nœuds qui sont contactés depuis le réseau qui opère la solution n'ont pas besoin d'avoir connaissance de la solution utilisée (et donc d'être compatibles ou configurés pour).
- *IPv6 natives* : Oui si les IPv6 de sortie n'utilisent pas un préfixe particulier qui désigne la solution mise en œuvre.
- *Universelle* : Oui si la solution fonctionne pour la couche IP sans distinction du type de trafic, non si elle répond à des besoins applicatifs spécifiques.

5.6 Conclusion

La solution du NAT64 semble la plus optimale en terme de charge de travail pour l'administrateur, puisque c'est une solution totalement transparente sans encapsulation, qui concerne tout le réseau à la fois, et qui ne nécessite pas de double adressage. Le NAT64 pouvant être configuré de façon à ne pas nécessiter de lien IPv6, il répond à tous les cas d'utilisation. Puisque le réseau sous-jacent ne nécessite pas de double pile, le réseau peut être entièrement géré en IPv6 et profiter pleinement des nouveautés, *a minima* en interne.

Il est donc le meilleur moyen de préparer son réseau pour l'avenir : il pourra commencer sans lien IPv6, continuer avec un lien IPv6 tout en gardant une compatibilité IPv4, et enfin terminer en remplaçant la machine du NAT64 par un simple pare-feu, sans quasiment ne faire aucun changement sur le réseau.

La figure 1.4 présentée dans la section *IPv6 day*, page 11, indique la régression de l'utilisation des méthodes alternatives comme le 6to4, au profit d'IPv6 natives (éventuellement en utilisant du 6rd, de l'ISATAP ou du NAT64).

6

Routage

« *Efforts and courage are not enough without purpose and direction.* » [1] - John F. Kennedy

6.1 Généralités

Il y a peu de différences entre le routage IPv4 et le routage IPv6, sauf que celui-ci apporte de nouvelles possibilités en terme de routage dynamique.

Le choix du chemin se fait toujours selon la règle du *more longest*, c'est à dire la route la plus précise.

Sur du matériel Cisco, le routage entre interfaces s'active pour l'unicast (fonctionnalité de routage classique) et le multicast (inutile pour les adresses multicast de lien local qui ne sont pas routées) :

```
Routeur(config)# ipv6 unicast-routing
Routeur(config)# ipv6 multicast-routing
```

Ce document n'ayant pas pour vocation de former à la création d'un opérateur, la partie concernant le routage dynamique sera succincte.

6.2 Statique

Aucune différence du côté du routage statique, pour lequel les outils ont tous été adaptés à l'identique. À noter que la convention voudrait que les routeurs soient toujours indiqués par une adresse de lien local.

Ajouter une route statique avec une route par défaut et afficher la table (Cisco) :

1. http://www.presidency.ucsb.edu/ws/index.php?pid=74076

```
Routeur(config)# ipv6 route 2001:db8:c::/64 2001:db8:a::1
Routeur(config)# ipv6 route ::/0 2001:db8:c::1
Routeur# sh ipv6 route
```

Avec un système GNU/Linux :

```
# ip route add 2001:db8:c::/64 2001:db8:a::1
# ip route add default via 2001:db8:c::1
# ip -6 route
```

6.3 Dynamique

6.3.1 Correction dynamique des chemins

La principale nouveauté concerne l'élaboration dynamique et participative du chemin le plus court.

Dans une configuration classique, une machine U1 décidera d'envoyer ses paquets à destination de U2 en passant par sa passerelle par défaut R1, si U2 ne fait partie d'aucun des réseaux auxquels il est directement relié (figure 6.1 page 76).

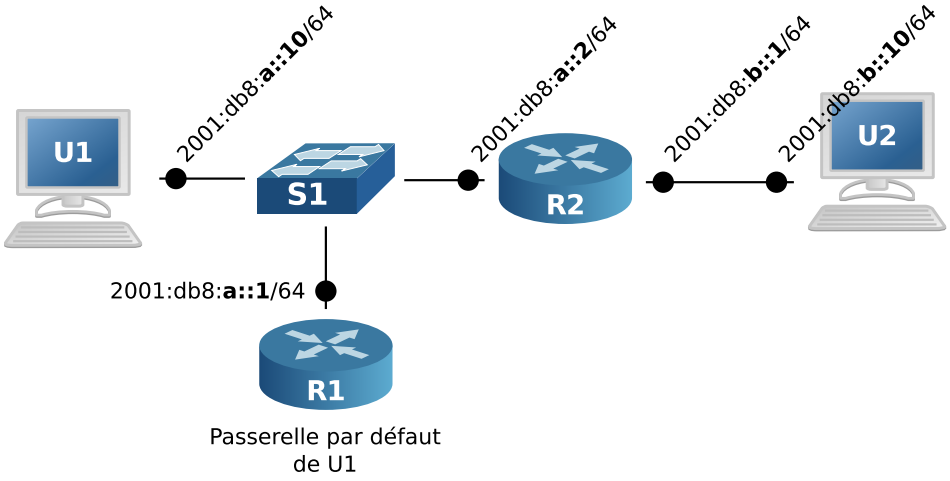


Figure 6.1 – *Ajout dynamique d'une route plus optimisée.*

Mais si R1 décide d'après ses règles de routage de transmettre les paquets à un routeur R2 qui est sur le même lien (cas illustré par le schéma ci-dessus), la situation révèle que U1 aurait pu les transmettre directement à R2 sans passer par R1, et donc qu'il lui manque une route.

R1 transmettra les paquets à R2, mais délivrera aussi un message ICMPv6 de type *redirect*, pour annoncer la route de R2 à U1. Si celui-ci la prend en compte, les prochains paquets à destination de U2 passeront directement par la nouvelle règle de routage.

6.3.2 Protocoles

La plupart des protocoles de routage dynamique ont évolué de façon à s'abstraire de la couche 3, pour ne pas avoir à différencier un trafic pour IPv4 d'un trafic pour IPv6 :

RIPng : Basée sur le protocole RIPv2 (RFC 2453), la version RIPng (RFC 2080 et 2081) supporte l'IPv6. Il utilise le port UDP 521 au lieu du port 520, différencie la table de routage IPv4 de la table IPv6, utilise des adresses de lien local pour le *next-hop*, transmet les préfixes et met à disposition l'adresse multicast `ff02::9` correspondant à *all-RIP-routers*. Les authentifications sont basées sur IPsec.

OSPFv3 : Contrairement à sa version précédente, la version 3 (RFC 5340) est indépendante du protocole réseau, ce qui lui permet de supporter parfaitement l'IPv6. Le *RouterID* (sur 32 bits) sert à présent à identifier directement le routeur, puisque les LSA Router et Network State ne contiennent plus d'adresses de réseau, et doit donc être fixé manuellement (ou utiliser l'adresse IPv4). Le *next-header* correspondant à OSPFv6 est 89 et comme RIPng l'authentification profite de la disponibilité systématique d'IPsec. De la même façon aussi, il utilise de préférence les adresses de lien local. L'adresse multicast `ff02::6` correspond à *OSPFv3 AllDR routers*.

IS-IS :[2] Contrairement à l'OSPF, IS-IS n'a pas attendu pour opérer en couche 2 puisqu'il observe ce comportement depuis toujours. Il a donc toujours été opérationnel pour l'IPv6. Seuls trois types d'information nouveaux sont apparus : *IPv6 Reachability TLV* pour annoncer les routes avec leur préfixe et leur métrique, *IPv6 Interface Address TLV* pour transmettre les adresses de type lien local et *IPv6 NLPID* pour annoncer le support de l'IPv6.

BGP4+ : Cette nouvelle version de BGP (RFC 2545) se base sur MBGP (*Multiprotocol BGP*, RFC 4760) et BGP4 (RFC 4271). Comme pour OSPFv3 le *RouterID* doit être fixé manuellement ou utiliser une adresse IPv4.

La figure 6.2 page 78 apporte une représentation graphique des échanges BGP IPv6 entres les AS enregistrés dans le *Looking Glass*[3] du Ring (projet de mise à disposition de machines virtuelle au sein des AS participants, pour permettre aux confrères de résoudre un problème de routage depuis l'intérieur des AS distants) du NLNog[4] (proposée par Job Snijders[5] et reportée par Jérôme Nicolle sur la liste du FRnOG).

Elle tend à prouver que l'Internet IPv6 est déjà bien en place et que le réseau est déjà loin de la chimère qui est parfois dénoncée.

6.4 Préfixe spécial DDoS

La RFC 6666 publiée fin août 2012[6] prévoit que le préfixe `0100::/64` est désormais réservé à la redirection du trafic potentiellement lié à un DDoS (*Distributed Deny of Service*).

3. http://lg.ring.nlnog.net/summary/lg01/ipv6
4. Groupe d'opérateurs réseaux néerlandais : http://nlnog.net.
5. Version haute résolution : http://instituut.net/~job/ring-lg/ipv6-bgp-adjacencies.jpeg (version IPv4 : http://instituut.net/~job/ring-lg/ipv4-bgp-adjacencies2.png).
6. Merci à Stéphane Bortzmeyer, qui a annoncé la nouvelle sur la liste FRnOG le 21 août, et qui a publié un article sur son blog, duquel cette section est inspirée : http://www.bortzmeyer.org/6666.html

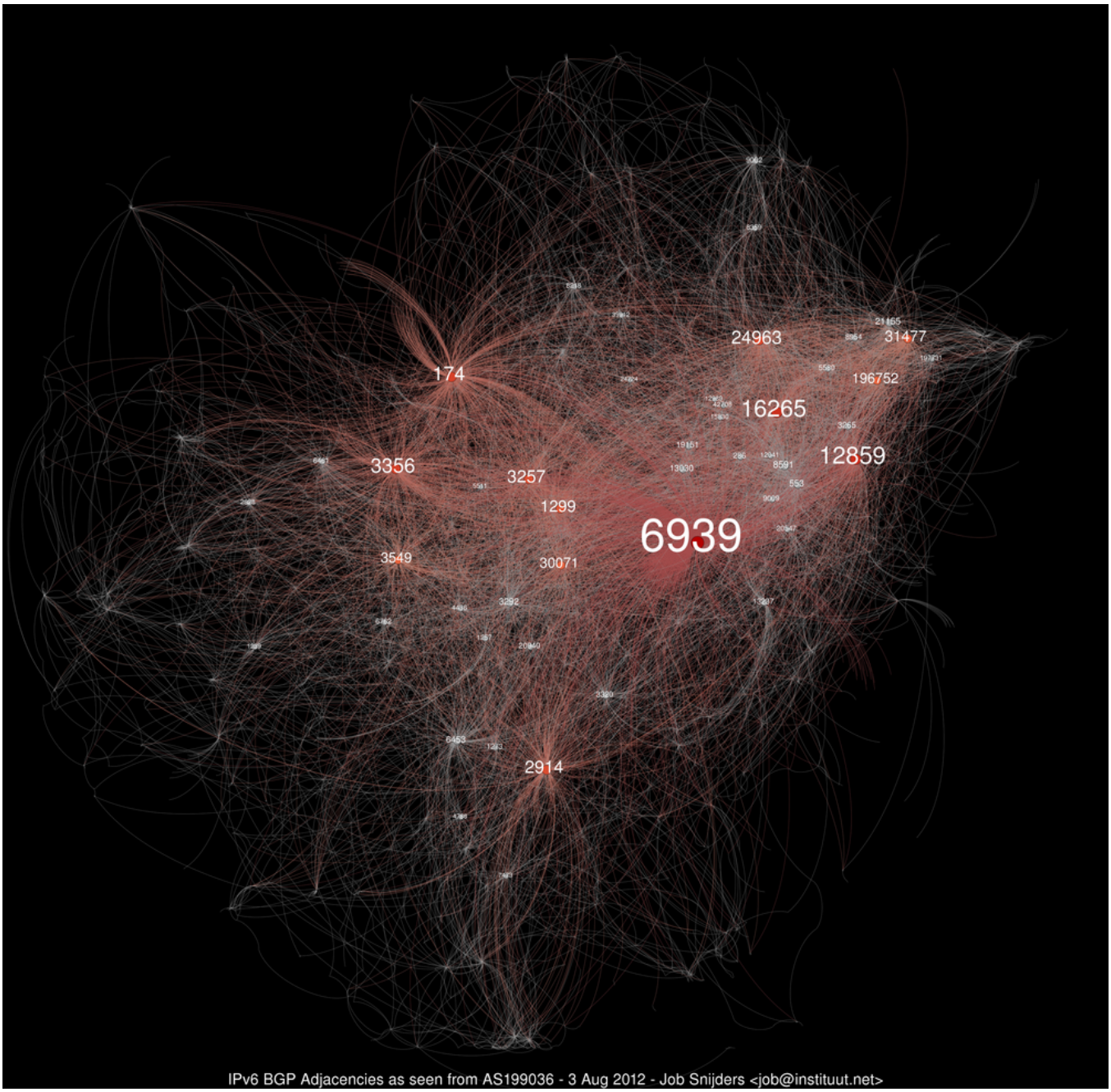


Figure 6.2 – *Échanges BGP en IPv6 enregistrés par le Ring du NLNOG.*

En cas d'attaque de ce type, une technique couramment utilisée est le RTBH (*Remote Triggered Black Hole*, RFC 5635 et 3882), souvent raccourcie en *black hole*. La victime demande alors à son FAI de rejeter tous les paquets provenant d'une plage précise d'adresses IP, éventuellement en précisant une liste de ports. Ce dernier s'exécute en configurant ses règles de routage dynamique, pour rediriger les paquets incriminés.

Les paquets peuvent être totalement oubliés, ou redirigés dans un tunnel qui aboutira sur un dispositif d'analyse qui décidera de leur sort. Les adresses utilisées pour la redirection sont parfois des adresses privées, ou issues de la plage des adresses de documentation. Cette façon de faire n'étant pas satisfaisante, la nouvelle plage d'IP, désormais réservée, est destinée à servir pour ces redirections.

Les adresses IP de la plage `0100::/64` indiquent donc clairement un trafic susceptible de venir d'un DDoS, et ne doivent pas être routées en dehors de l'infrastructure de l'opérateur, au risque sinon de propager l'attaque.

7

Mobilité

« *I'm very excited about having the Internet in my den.* » [1] - Steve Jobs (1994)

7.1 Généralités

Le concept de mobilité s'applique lorsqu'un mobile (un ordiphone, une tablette, un ordinateur, etc.) change d'adresse IP en changeant de réseau (wifi, GSM, etc.), tout en souhaitant rester joignable en permanence via l'adresse connue de son équipement, sur son réseau personnel.

S'il ne change pas d'adresse en changeant de réseau, comme c'est le cas pour le *roaming* en wifi, le mécanisme ne concerne pas la couche IP.

Cette notion n'est pas une nouveauté puisqu'elle existait déjà en IPv4 (RFC 2002), dans une version qui n'était pas très exploitable étant donnée la complexité des réseaux IPv4 constitués de NAT et le manque d'adresses disponibles. La version IPv6 (RFC 6275) résout ces problèmes, et apporte plus de légèreté, moins de prérequis et plus de sécurité.

Dans les deux modes existants, le nœud qui souhaite utiliser le service de mobilité possédera deux adresses :

- **home address (HoA) :** L'IPv6 de la machine qui est connue de tous, et qui appartient donc au préfixe du LAN auquel il appartient initialement (maison ou bureau, selon les cas).
- **care-of address (CoA) :** L'IPv6 de mobilité, qui change selon les réseaux auxquels la machine se connecte lors de son déplacement, imprévisible.

1. http://the99percent.com/articles/7044/9-Awesome-Interviews-with-Creative-Visionaries

Nous parlerons, conformément à la littérature, de mobile (*mobile node* - MN) pour le nœud qui se déplace, et de correspondant (*correspondent node* - CN) pour celui qui tente de joindre le mobile. Lorsqu'il est sur son réseau initial, le routeur qui gère celui-ci est désigné comme *home agent* (HA) sinon c'est le routeur d'accès (*router access* - RA).

7.2 Sans optimisation du chemin (tunnel bidirectionnel)

Ce mode correspond à celui par défaut, puisqu'il ne nécessite pas de configuration du côté du correspondant, qui ignorera totalement les déplacements du mobile. Il tentera donc toujours de le contacter avec sa *home address*, comme si le mobile restait en permanence dans son réseau.

Lorsque le mobile est dans son réseau initial (figure 7.1 page 82), le correspondant peut le contacter normalement en utilisant son adresse connue et en passant par le *home agent* par les mécanismes de routage habituels.

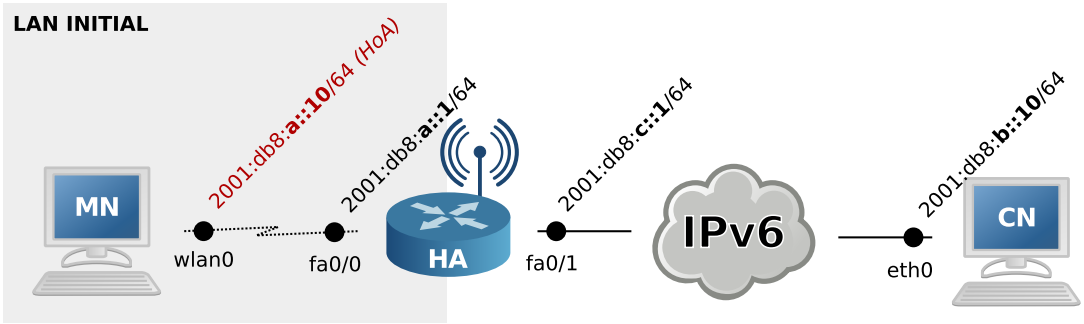


Figure 7.1 – *Réseau de référence du mobile.*

Dès lors que le mobile est en déplacement (figure 7.2 page 83), le correspondant continue de contacter le *home agent* pour atteindre le mobile, il faut donc qu'il soit capable de transmettre les paquets à celui-ci où qu'il soit.

Pour ce faire, dès lors que le mobile acquiert une adresse (par DHCP ou autoconfiguration *stateless*) qui ne correspond pas à sa *home address*, il doit prévenir son *home agent* de l'endroit où il se situe pour se faire livrer ses paquets (échanges en IPsec pour éviter les usurpations d'identité), comme illustré en figure 7.3 page 83.

Ainsi, lorsque le correspondant souhaitera contacter le mobile, le *home agent* se chargera de transmettre les paquets à l'adresse *care-of* (qui délivrera ensuite en interne le paquet à l'interface qui a gardé la *home address*) reçue dernièrement par le mobile au travers du tunnel établi (figure 7.4 page 83).

Le tunnel étant bidirectionnel, les réponses à destination du correspondant l'emprunteront aussi.

Cette méthode ne nécessite qu'une compatibilité au niveau des équipements du mobile (routeur et poste utilisateur), mais elle n'est pas optimale en terme de chemins. De plus, l'utilisation d'un tunnel alourdit le processus et peut poser problème dans le cas d'application sensibles comme la téléphonie.

Pour ces raisons, une seconde méthode existe.

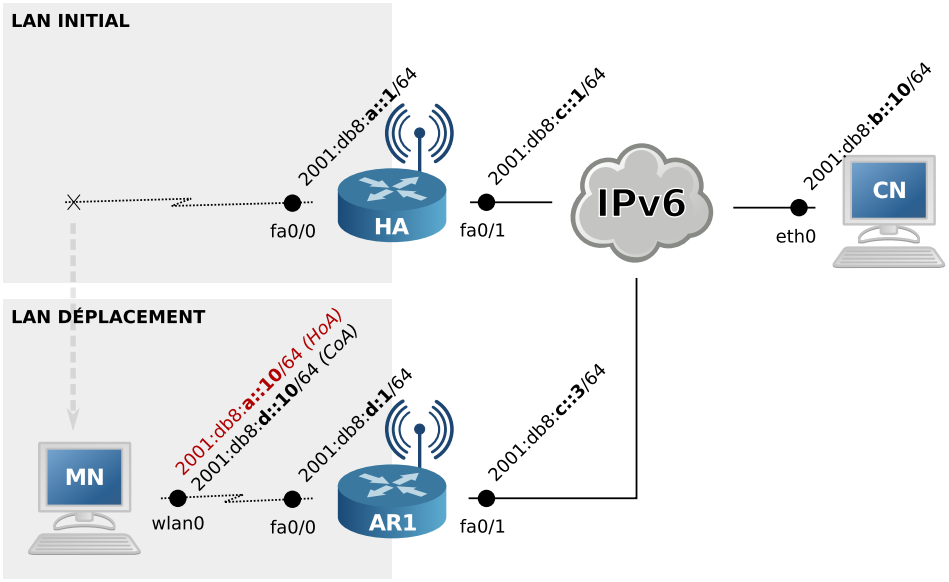


Figure 7.2 – *Mobile en déplacement.*

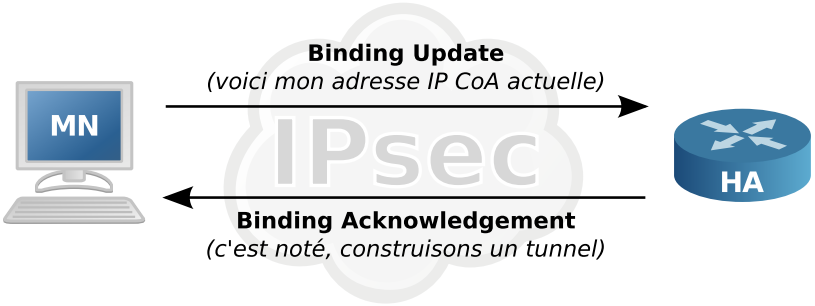


Figure 7.3 – *Mise à jour de l'adresse* care-of *du mobile.*

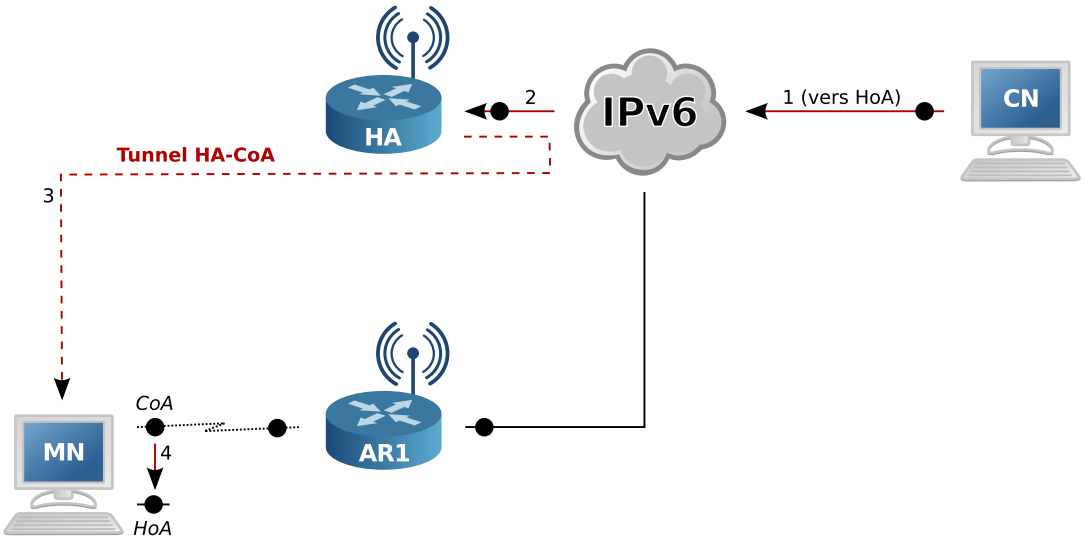


Figure 7.4 – *Contact du mobile au travers de son* home agent.

7.3 Avec optimisation des chemins

Cette méthode permet au correspondant de contacter directement le mobile en ayant connaissance de son adresse *care-of*.

Le mobile commence par créer un tunnel avec son *home agent* de la même façon que dans la premier mode. Puis, afin d'assurer la sécurité de ce système, le mobile doit avertir le correspondant de son adresse *care-of* actuelle en utilisant un mécanisme dédié à cet usage. Le mécanisme est illustré en figure 7.5 page 84.

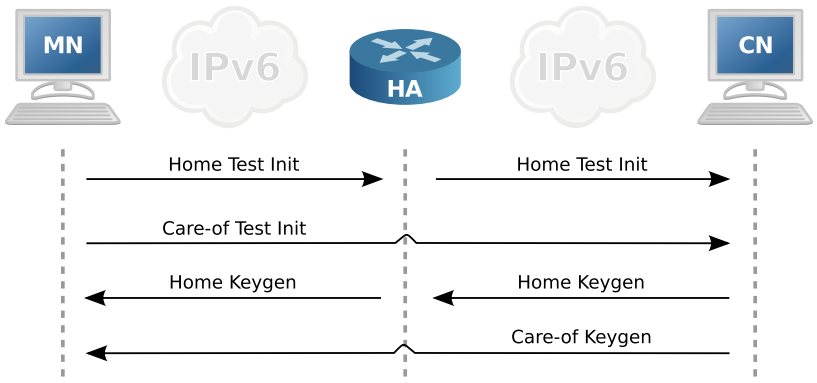


Figure 7.5 – *Mobilité avec optimisation des chemins.*

En passant à la fois par le tunnel et le chemin direct, le correspondant peut transmettre des clés qui permettront de s'assurer que le mobile est bien celui qu'il prétend être. Ainsi, les deux jetons (*keygen*) reçus seront utilisés par le mobile pour créer une clé de 20 octets qui servira à chiffrer les messages *Binding Update*. Ils serviront par la suite à indiquer au correspondant un déplacement (et donc une nouvelle adresse *care-of*). Les paquets qui suivront iront directement de la *care-of* à l'adresse du correspondant et inversement, sans souffrir de la lourdeur d'un tunnel.

La procédure d'authentification échoue (probablement parce que le correspondant ne supporte pas les fonctionnalités de mobilité), la communication se fera par le tunnel, dans le mode précédent.

Puisqu'il peut y avoir des paquets perdus le temps de la réception du message *Binding Update*, un mécanisme de *Fast Handover* est décrit dans les RFC 5568 et 4260. Plusieurs *home agents* peuvent aussi être utilisés pour assurer sa disponibilité.

7.4 Autres solutions

Deux autres solutions pour faire de la mobilité IPv6 existent, l'une étant de l'ordre de l'optimisation et l'autre dans un esprit différent :

HMIPv6 : Il s'agit d'une extension (RFC 4140) de la mobilité IPv6 vue plus haut, consistant à imposer une organisation hiérarchique au moment de la transmission des messages de mise à jour (*Binding Update*). Ainsi, ceux-ci seront systématiquement envoyés à un nouvel élément appelé *Mobility*

Anchor Point (MAP) qui se chargera la première fois d'en rediffuser à tous les correspondants. Ces nœuds sont communiqués via les *Router Advertisements* distribués par le réseau sur lequel se trouve le mobile. Lorsque le mobile changera de réseau, ce système parie sur le fait qu'il recevra plusieurs fois le même nœud MAP à proximité, qu'il n'aura alors qu'à prévenir de son changement de réseau. Du côté des correspondants, ils continuent à communiquer avec le nœud MAP, sans qu'aucun changement ne leur soit communiqué. Dans ce cas, les messages de mise à jour ne sont donc envoyés qu'en un seul exemplaire et à un nœud topologiquement proche, plutôt qu'aux *n* correspondants éparpillés.

NEMO : Son fonctionnement ressemble beaucoup au premier mode avec le tunnel (la version optimisation de routes n'existe pas encore), mais il concerne le réseau dans son intégralité plutôt que les postes utilisateurs séparément. Ainsi, c'est le routeur (*Mobile Router*) qui se charge du travail, en diffusant des routes dynamiques. Puisqu'il permet de déplacer un réseau en entier, ce mécanisme (RFC 3963) permet aussi d'apporter de la mobilité à des postes utilisateurs qui ne le supportent pas.

7.5 Compatibilité

La plupart des RFC citées ci-dessus sont actuellement marquées *PROPOSED STANDARD*. Ces fonctionnalités n'ont donc pas encore fait l'objet d'un intérêt démentiel, et ce retard se retrouve au niveau des solutions logicielles existantes.

Il fut un temps où le site *mobile-ipv6.org* était la référence dans ce domaine. Ainsi, la plupart des documentations s'y réfèrent pour trouver les ressources nécessaires, alors que le site semble avoir disparu depuis quelques années. On trouve également beaucoup de références au projet Nautilus6 [2], qui fut un groupe de travail sur ces questions jusqu'à 2008, et qui a produit quelques solutions logicielles qui semblent condamnées à devenir obsolètes. Deux solutions, toutes deux basées sur le projet MIPL (apparemment autrefois délivré par *mobile-ipv6.org*) : l'une supportée par le projet USAGI [3] et l'autre par UMIP.org. Ces deux solutions semblent être uniquement destinées à GNU/Linux et BSD [4] et permettent à un nœud de devenir n'importe lequel des composants qui interviennent dans la mobilité. La solution de UMIP.org semble être la plus à jour et la plus fonctionnelle. À noter que CentOS propose un paquet *mipv6-deamon* dans son gestionnaire de paquets.

Du côté de Windows, c'est encore plus drôle : Windows XP et Server 2003 supportaient la mobilité IPv6 [5] (uniquement pour le mode correspondant [6]) mais ils ont jugé que c'était trop compliqué de l'inclure dans Vista [7]. Par conséquent la fonctionnalité semble totalement abandonnée dans les nouvelles versions, marquant ainsi une régression.

Enfin pour Mac, il ne semble n'y avoir qu'un port de SHISA [8], l'un des projets de Nautilus6 qui semble à la fois pauvre en fonctionnalités et plus ou moins à l'abandon.

2. http://www.nautilus6.org
3. http://www.linux-ipv6.org
4. http://devjlanza.wordpress.com/2011/09/11/mobile-ipv6-user-space-daemons-in-linux-ubuntu/
5. http://www.microsoft.com/en-us/download/details.aspx?id=10905
6. http://www.seattlepro.com/uw/docs/IPv6/ipv6_faq.htm
7. http://blogs.technet.com/b/ipv6/archive/2007/05/08/mobile-ipv6.aspx
8. http://www.momose.org/macosx/mip6.html

Au niveau des équipements réseaux, le constat est plus encourageant puisque Cisco supporte les fonctionnalités de *home agent* sur de nombreux modèles de routeurs.

Une tentative d'expérimentation du premier mode avec un routeur Cisco, un mobile Debian et un correspondant Mac, est disponible dans la section 8.7 page 109.

8

Expérimentations

« *Talk is cheap. Show me the code.* » [1] - Linus Torvalds (2000)

« *I don't care if it works on your machine ! We are not shipping your machine !* » - Vidiu Platon

8.1 Généralités

Les expérimentations ont été effectuées à l'aide de :

- 1 MacBookPro (U2) : *Max OS X Darwin*
- 1 PC Dell Latitude E4300 (U1) : *GNU/Linux Debian Wheezy i386*
- 1 serveur Dell PowerEdge 2950 : *GNU/Linux Debian Squeeze i386*
- 1 switch Cisco 2960 : *C2960 Software (C2960-LANBASE-M), Version 12.2(25)SEE2, RELEASE SOFTWARE (fc1)*
- 1 firewall Cisco ASA 5505 : *Cisco Adaptive Security Appliance Software Version 9.0(0)11* (prêté par Cisco/Axians, en version bêta)
- 2 AP wifi Cisco Aironet 1130AG (autonomes) : *C1130 Software (C1130-K9W7-M), Version 12.4(10b)JA3, RELEASE SOFTWARE (fc1)*
- 5 routeurs Cisco 2811 : *2800 Software (C2800NM-ADVIPSERVICESK9-M), Version 12.4(24)T6, RELEASE SOFTWARE (fc2)*

Tous les équipements réseau ont été réinitialisés avant chaque expérimentation.

1. Talkischeap.Showmethecode.

Les plans d'adressage utilisent les plages d'adresses IP ainsi que les TLD destinés à la documentation :

- **RFC 5737 :** `192.0.2.0/24` (TEST-NET-1) et `198.51.100.0/24` (TEST-NET-2)
- **RFC 3849 :** `2001:DB8::/32`
- **RFC 2606 :** `.example`

Elles sont routables mais ne doivent pas utilisées en production.

Ces plans d'adressage sont conformes aux schémas destinés à expliquer les protocoles, disponibles dans la section qui leur correspond. Les nuages ont parfois été remplacés par un routeur pour simuler le réseau Internet. Les adresses n'utilisent jamais les EUI-64 pour une raison de clarté et pour une gestion manuelle plus aisée.

La photo en figure 8.1 page 88 donne un aperçu de l'installation avec les différents équipements.

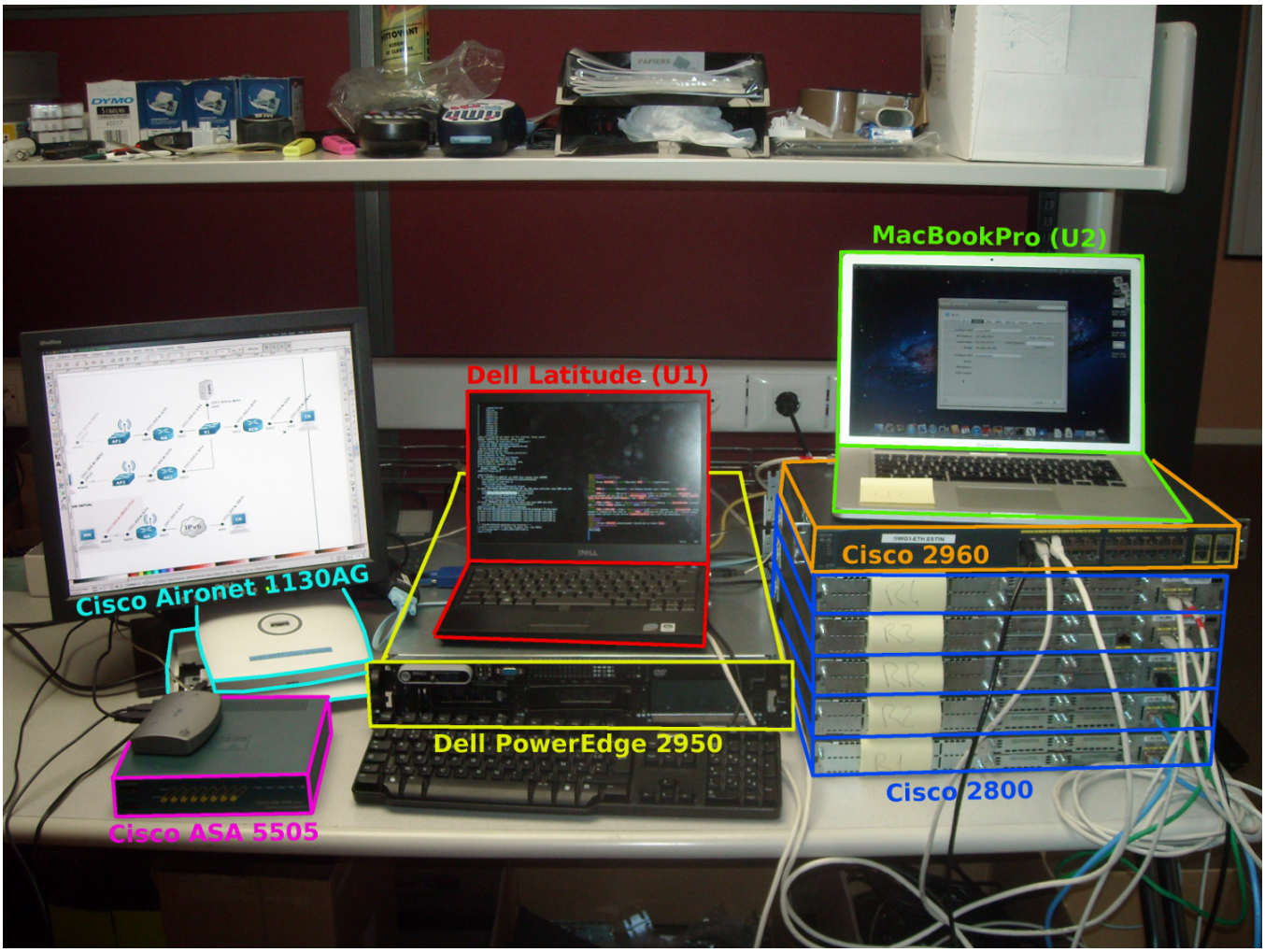


Figure 8.1 – *Équipements utilisés pour les tests en laboratoire.*

8.2 Autoconfiguration *stateless* (avec DNS) via un routeur GNU/Linu

Introduction

Le DNS n'étant pas prévu initialement pour l'autoconfiguration *stateless*, celle-ci est souvent décriée puisqu'elle nécessite dans tous les cas un DHCP pour effectuer cette tâche.

L'objectif est donc de tester les facilités de l'autoconfiguration, en prenant en compte le RDNSS, qui permet depuis récemment de diffuser des adresses DNS via les *Router Advertisements*. Une machine dépourvue de toute configuration réseau particulière devra réussir à parfaitement s'insérer (obtenir une adresse IP et un serveur DNS) dans un réseau IPv6, sans utiliser de DHCP.

Expérimentation

La machine utilisateur U1 étant une Debian, l'autoconfiguration *stateless* du poste est activée par défaut, comme sur la plupart des systèmes.

La topologie et le plan d'adressage sont décrits dans la figure 8.2 page 89.

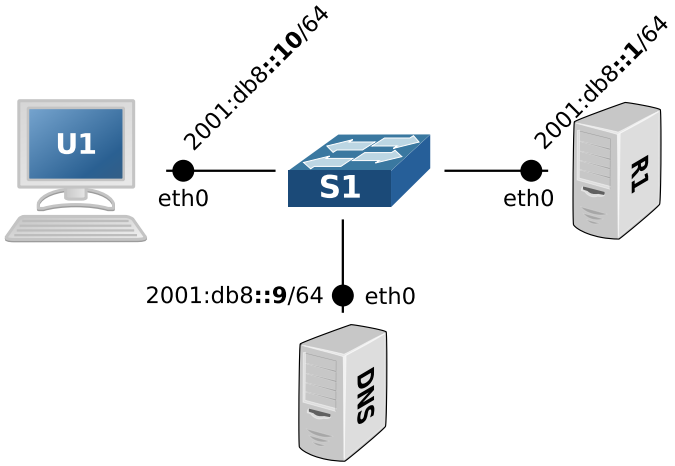


Figure 8.2 – *Installation liée à l'expérimentation de l'autoconfiguration* stateless *(SLAAC).*

Configuration du routeur R1 :

```
R1# apt-get install radvd
```

Configuration de Radvd (*/etc/radvd.conf*) :

```
interface eth0
{
        AdvSendAdvert on;
        prefix 2001:db8::/64
        {
        };
        RDNSS 2001:db8::9
```

```
8         {
9         };
10 };
```

Configuration de l'utilisateur U1 :

```
1 U1# apt-get install rdnssd
```

Patienter 10 minutes (par défaut le *MaxRtrAdvInterval* de Radvd est positionné à 600 secondes), ou relancer l'interface réseau.

Nouvelle adresse IPv6 :

```
1 U1# ip -6 addr show dev eth0
2 2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
3     inet6 2001:db8:0:0:221:70ff:feec:b49c/64 scope global dynamic
4        valid_lft 86336sec preferred_lft 14336sec
5     inet6 fe80::221:70ff:feec:b49c/64 scope link
6        valid_lft forever preferred_lft forever
```

Nouveau serveur DNS :

```
1 U1# cat /etc/resolv.conf
2 nameserver 2001:db8::9
```

Si d'autres serveurs étaient déjà configurés, il n'auront pas été écrasés.

Conclusion

Afin de diversifier les tests, mais aussi parce que Cisco ne semble pas du tout supporter le RDNSS, une machine GNU/Linux a été utilisée en tant que routeur. Malheureusement, cette option essentielle est encore trop récente pour pouvoir l'exploiter sur du matériel réseau classique, et continue donc de condamner ce mode d'autoconfiguration.

Du côté des clients, GNU/Linux semble aussi plus ou moins le seul à être en avance et à supporter parfaitement cette option. L'attribution des IP quant à elle, ne pose aucun problème, quelque soit les postes utilisateurs supportant l'IPv6 ou le matériel réseau.

Bien que l'option n'ait pas été exploitée dans cette expérimentation, Radvd supporte aussi le DNSSL depuis sa version 1.8 :

```
1 interface eth0 {
2   ...
3   DNSSL lan.local { };
4 }
```

8.3 Tunnel statique

Introduction

L'objectif de cette expérimentation consiste à tester la faisabilité de la création d'un tunnel statique IPv6 entre deux routeurs Cisco. Le tunnel devra faire permettre de faire communiquer deux réseaux IPv6 entre eux, en passant par un routeur qui ne supporte que l'IPv4 et qui ne doit pas nécessiter de configuration particulière.

Cette expérimentation permettra de comprendre le concept de la tunnelisation IPv6 over IPv4, pour tester ensuite les tunnels dynamiques avec le 6to4 ou le 6rd.

Expérimentation

La topologie et le plan d'adressage sont décrits dans la figure 8.3 page 91.

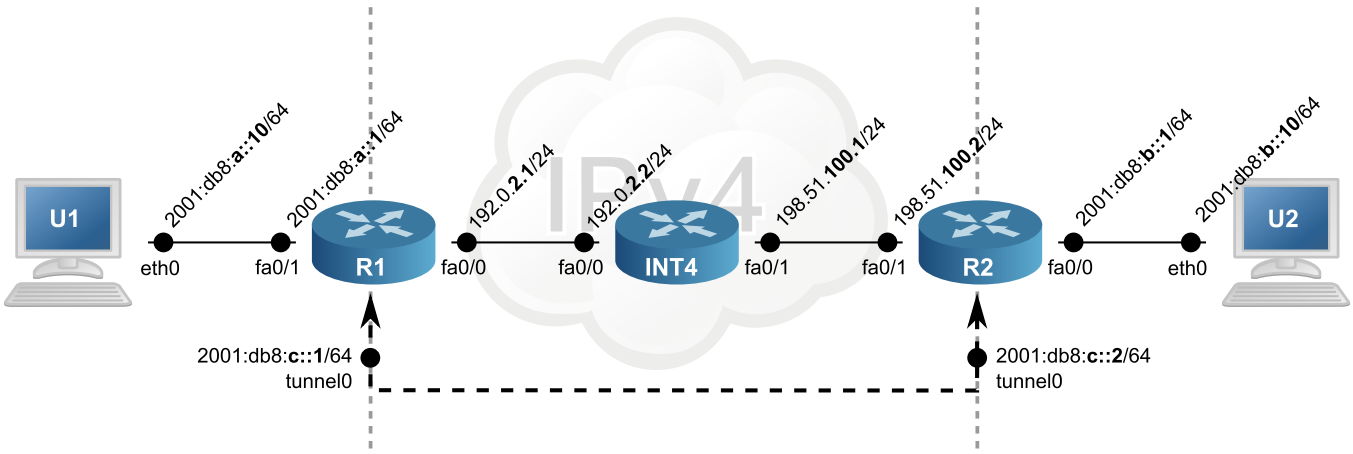


Figure 8.3 – *Installation liée à l'expérimentation des tunnels statiques.*

Configuration du routeur R1 :

```
R1> en
R1# conf t
R1(config)# int fa 0/1
R1(config-if)# ipv6 addr 2001:db8:a::1/64
R1(config-if)# no shut
R1(config-if)# int fa 0/0
R1(config-if)# ip addr 192.0.2.1 255.255.255.0
R1(config-if)# no shut
R1(config-if)# int tunnel 0
R1(config-if)# ipv6 addr 2001:db8:c::1/64
R1(config-if)# tunnel mode ipv6ip
R1(config-if)# tunnel source 192.0.2.1
R1(config-if)# tunnel dest 198.51.100.2
R1(config-if)# exit
```

```
R1(config)# ipv6 unicast-routing
R1(config)# ip route 0.0.0.0 0.0.0.0 192.0.2.2
R1(config)# ipv6 route ::/0 Tunnel 0
```

Configuration du routeur INT4 :

```
INT4> en
INT4# conf t
INT4(config)# int fa 0/0
INT4(config-if)# ip addr 192.0.2.2 255.255.255.0
INT4(config-if)# no shut
INT4(config-if)# int fa 0/1
INT4(config-if)# ip addr 198.51.100.1 255.255.255.0
INT4(config-if)# no shut
INT4(config-if)# exit
INT4(config)# ip routing
```

Configuration du routeur R2 :

```
R2> en
R2# conf t
R2(config)# int fa 0/1
R2(config-if)# ip addr 192.51.100.2 255.255.255.0
R2(config-if)# no shut
R2(config-if)# int fa 0/0
R2(config-if)# ipv6 addr 2001:db8:b::1/64
R2(config-if)# no shut
R2(config-if)# int tunnel 0
R2(config-if)# ipv6 addr 2001:db8:c::2/64
R2(config-if)# tunnel mode ipv6ip
R2(config-if)# tunnel source 198.51.100.2
R2(config-if)# tunnel dest 192.0.2.1
R2(config-if)# exit
R2(config)# ipv6 unicast-routing
R2(config)# ipv6 route ::/0 Tunnel 0
```

Configuration de l'utilisateur U1 :

```
U1# ip a a 2001:db8:a::10 dev eth0
```

Configuration de l'utilisateur U2 :

```
U2# ip a a 2001:db8:b::10 dev eth0
```

Vérification du tunnel sur R1 :

```
R1# sh ipv6 int brief
[...]
Tunnel0            [up/up]
```

Test de U1 à U2 :

```
U1$ ping6 fc00::2:2
PING fc00::2:2(fc00::2:2) 56 data bytes
64 bytes from fc00::2:2: icmp_seq=1 ttl=62 time=2.61 ms
64 bytes from fc00::2:2: icmp_seq=2 ttl=62 time=2.69 ms
64 bytes from fc00::2:2: icmp_seq=3 ttl=62 time=2.52 ms
```

Conclusion

En prenant la peine de configurer les deux routeurs aux extrémités des réseaux IPv6, la création du tunnel ne pose aucun problème, et permet de faire passer facilement de l'IPv6 au travers d'un réseau IPv4.

Un complément intéressant serait de tester avec des équipements de pare-feu dotés d'une configuration classique, pour constater les éventuels problèmes liés au protocole 41.

8.4 Tunnel 6to4 avec un relais

Introduction

L'expérimentation suivante vise à tester le bon fonctionnement et la facilité d'installation d'un relais 6to4. Deux réseaux uniquement IPv6 devront réussir à communiquer entre eux au travers d'un réseau qui ne gère que l'IPv4 et qui ne devra pas nécessiter de configuration spécifique.

Contrairement au tunnel statique, le routeur à l'extrémité de la machine à contacter ne doit pas nécessiter de configuration particulière. Un relais 6to4 doit donc être installé à l'intersection entre un routeur qui ne gère que l'IPv4 et un autre qui ne gère que l'IPv6. L'expérimentation visera également à tester l'adresse anycast normalisée pour ce type d'équipement.

Expérimentation

La topologie et le plan d'adressage sont décrits dans la figure 8.4 page 93.

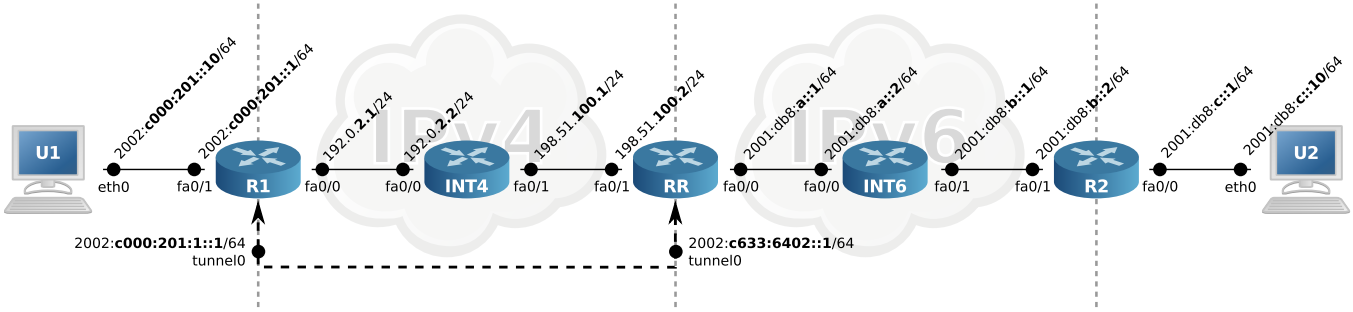


Figure 8.4 – *Installation liée à l'expérimentation du 6to4.*

Configuration du routeur R1 :

```
R1> en
R1# conf t
R1(config)# int fa 0/1
R1(config-if)# ipv6 addr 2002:c000:201::1/64
R1(config-if)# no shut
R1(config-if)# int fa 0/0
R1(config-if)# ip addr 192.0.2.1 255.255.255.0
R1(config-if)# no shut
R1(config-if)# int tunnel 0
R1(config-if)# ipv6 addr 2002:c000:201:1::1/64
R1(config-if)# tunnel source fa 0/0
R1(config-if)# tunnel mode ipv6ip 6to4
R1(config-if)# exit
R1(config)# ipv6 unicast-routing
R1(config)# ipv6 route 2002::/16 tunnel 0
R1(config)# ipv6 route ::/0 2002:c633:6402::1
```

Configuration du routeur INT4 :

```
INT4> en
INT4# conf t
INT4(config)# int fa 0/0
INT4(config-if)# ip addr 192.0.2.2 255.255.255.0
INT4(config-if)# no shut
INT4(config-if)# int fa 0/1
INT4(config-if)# ip addr 198.51.100.1 255.255.255.0
INT4(config-if)# no shut
INT4(config-if)# exit
INT4(config)# ip routing
```

Configuration du routeur RR :

```
RR> en
RR# conf t
RR(config)# int fa 0/1
RR(config-if)# ip addr 198.51.100.2 255.255.255.0
RR(config-if)# no shut
RR(config-if)# int fa 0/0
RR(config-if)# ipv6 addr 2001:db8:a::1/64
RR(config-if)# no shut
RR(config-if)# int tunnel 0
RR(config-if)# ipv6 addr 2002:c633:6402::1/64
RR(config-if)# tunnel source fa 0/1
RR(config-if)# tunnel mode ipv6ip 6to4
RR(config-if)# exit
RR(config)# ipv6 unicast-routing
RR(config)# ipv6 route 2002::/16 tunnel 0
RR(config)# ipv6 route ::/0 2001:db8:a::2
```

Contrairement aux tunnels statiques, il n'y a pas de `tunnel destination`, l'adresse étant déduite de l'IPv6 vers laquelle le paquet sera transmis.

Configuration du routeur INT6 :

```
RR> en
RR# conf t
RR(config)# int fa 0/0
RR(config-if)# ipv6 addr 2001:db8:a::2/64
RR(config-if)# no shut
RR(config-if)# int fa 0/1
RR(config-if)# ipv6 addr 2001:db8:b::1/64
RR(config-if)# no shut
RR(config-if)# exit
RR(config)# ipv6 unicast-routing
RR(config)# ipv6 route 2002::/16 2001:db8:a::1
RR(config)# ipv6 route ::/0 2001:db8:b::2
```

Configuration du routeur R2 :

```
R2> en
R2# conf t
R2(config)# int fa 0/1
R2(config-if)# ipv6 addr 2001:db8:b::2/64
R2(config-if)# no shut
R2(config-if)# int fa 0/0
R2(config-if)# ipv6 addr 2001:db8:c::1/64
R2(config-if)# no shut
R2(config-if)# exit
R2(config)# ipv6 unicast-routing
R2(config)# ipv6 route ::/0 2001:db8:b::1
```

Configuration de l'utilisateur U1 :

```
U1# ip a a 2002:c000:201::10/64 dev eth0
U1# ip -6 r a default via 2002:c000:201::1/64 dev eth0
```

Configuration de l'utilisateur U2 :

```
U2# ip a a 2001:db8:c::10/64 dev eth0
U2# ip -6 r a default via 2001:db8:c::1/64 dev eth0
```

Test de U1 à U2 :

```
U1$ ping6 2001:db8:c::10
PING 2001:db8:c::10(2001:db8:c::10) 56 data bytes
64 bytes from 2001:db8:c::10: icmp_seq=1 ttl=60 time=214 ms
64 bytes from 2001:db8:c::10: icmp_seq=2 ttl=60 time=3.83 ms
64 bytes from 2001:db8:c::10: icmp_seq=3 ttl=60 time=3.92 ms
```

Le routeur relais étant destiné à annoncer des routes en 2002::/16 (et qui ne peuvent être réduites), il devrait rejoindre les autres routeurs relais en utilisant l'adresse anycast appropriée.

Modification de RR pour lui attribuer l'adresse anycast normalisée en IPv4 (ainsi que son équivalent IPv6) :

```
RR(config)# int fa 0/1
RR(config-if)# no ip addr
RR(config-if)# ip addr 192.88.99.1 255.255.255.0
RR(config-if)# int tunnel 0
RR(config-if)# no ipv6 addr
RR(config-if)# ipv6 addr 2002:c058:6301::/48
%Tunnel0: Warning: 2002:C058:6301::/48 is a Subnet Router Anycast
```

Modification de INT4 pour lui ajouter une route IPv4 par défaut (RR n'appartient plus au même sous-réseau que lui) :

```
INT4(config)# ip route 0.0.0.0 0.0.0.0 fa 0/1
```

Modification de R1 pour qu'il contacte l'adresse anycast des relais 6to4 (s'il recevait des routes dynamiquement, ce ne serait donc pas obligatoirement le nôtre) :

```
R1(config)# no ipv6 route ::/0
R1(config)# ipv6 route ::/0 2002:c058:6301::
```

Nouveau test de U1 à U2 :

```
U1$ ping6 2001:db8:c::10
PING 2001:db8:c::10(2001:db8:c::10) 56 data bytes
64 bytes from 2001:db8:c::10: icmp_seq=3 ttl=60 time=3.84 ms
64 bytes from 2001:db8:c::10: icmp_seq=4 ttl=60 time=3.84 ms
64 bytes from 2001:db8:c::10: icmp_seq=5 ttl=60 time=3.88 ms
```

Conclusion

Un équipement peu coûteux comme un routeur Cisco 2800 permet de facilement mettre en place un routeur 6to4 avec un relais vers un réseau IPv6 natif.

L'adresse anycast ne pose aucun problème, et la machine U1 a pu contacter U2, sans que celle-ci ait connaissance de l'incapacité du réseau du U1 à communiquer en IPv6.

8.5 Tunnel 6rd avec relais

Introduction

Les tunnels 6rd sont proches des tunnels 6to4, mais nécessitent des configurations particulières, comme la diffusion des préfixes personnalisés ou le nombre de bits utiles pour représenter les adresses IPv4.

L'absence de maîtrise des relais 6to4 posant beaucoup de problème, l'objectif est de constater si son successeur est aussi simple à mettre en place, en permettant à deux machines de communiquer de

la même façon que pour le 6to4. Une machine dans le domaine 6rd est ajoutée pour vérifier que la configuration permet aussi de ne pas passer par le relais.

Expérimentation

La topologie et le plan d'adressage sont décrits dans la figure 8.5 page 97.

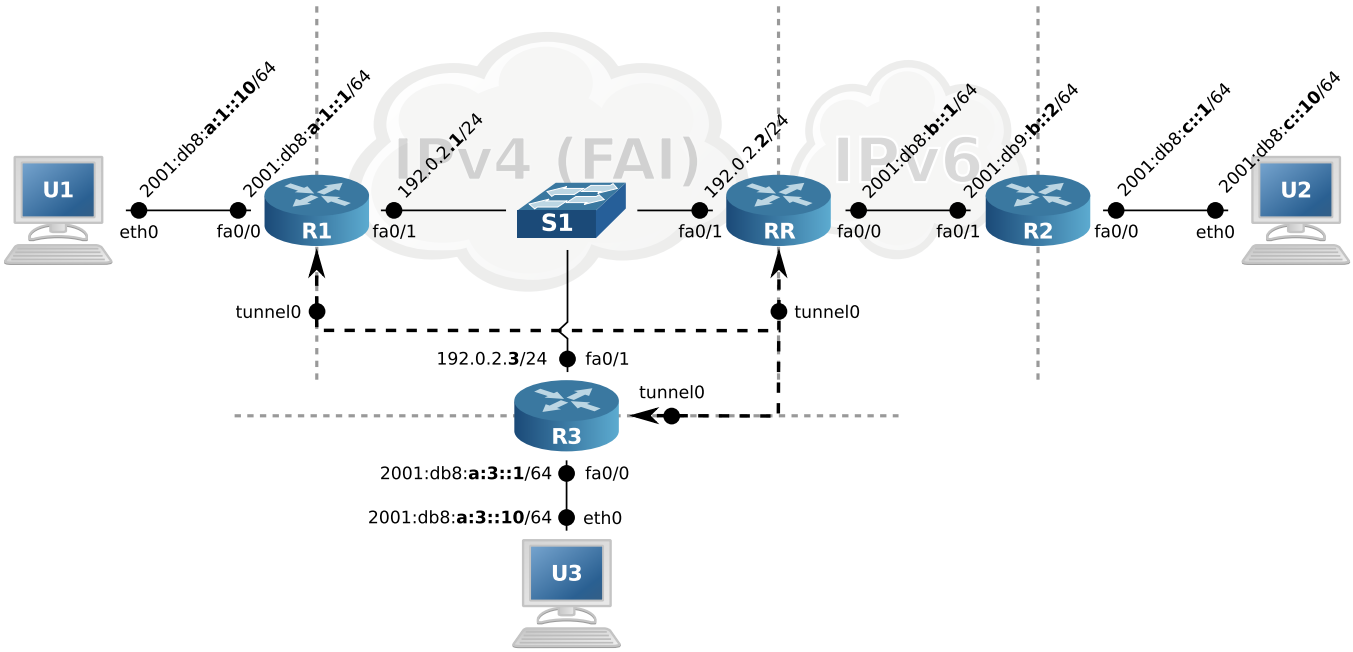


Figure 8.5 – *Installation liée à l'expérimentation du 6rd.*

Remarques :

- Pour simplifier la compréhension (et limiter le nombre de machines à utiliser), les réseaux Internet n'ont pas été simulés.
- On considère que les préfixes `2001:db8::/32` et `192.0.2.1/24` ont été alloués au FAI.
- Contrairement au 6to4, une machine a été ajoutée pour tester les connexions au sein du domaine.

Après avoir mis en place la topologie conçue, un problème de version des IOS s'est posé : contrairement au 6to4, le 6rd est encore trop récent et trop destiné à de gros matériels situés en cœur de réseau. Cette technologie n'est donc pas disponible sur les machines qui sont à disposition pour ces expérimentations. Pour connaître la liste des plate-formes supportées, se reporter au *Cisco Features Navigator*[2].

Voici la configuration du tunnel pour R1, correspondant à la conception (non-testée) :

```
1 R1> en
2 R1# conf t
3 R1(config)# int tunnel 0
4 R1(config-if)# tunnel source 192.0.2.1
```

2. http://tools.cisco.com/ITDIT/CFN/jsp/by-feature.jsp (*By feature* > *Search for 6rd* > *IP Tunneling - 6RD IPv6 Rapid Deployment* > *Add* > *Continue* > *Platform*).

```
R1(config-if)# tunnel mode ipv6ip 6rd
R1(config-if)# tunnel 6rd prefix 2001:db8:a::/48
R1(config-if)# tunnel 6rd ipv4 prefix-length 24
R1(config-if)# exit
R1(config)# ipv6 route 2001:db8:a::/48 Tunnel 0
R1(config)# ipv6 route ::/0 2001:db8:a:2::
```

Explications :

- On considère que le sous-réseau `2001:db8:a::/48` a été retenu par le FAI pour servir dans son domaine 6rd.
- Puisque toutes ses adresses IPv4 partagent le préfixe `192.0.2.0/24`, il n'a besoin que du dernier octet pour les différencier.
- Contrairement au 6to4, les adresses IPv6 des tunnels ne sont plus explicitement précisées et la destination est une adresse anycast du domaine.

Conclusion

Cette technologie est malheureusement encore trop récente, et destinée aux équipement très fortement orientés fournisseur d'accès à Internet. Les rares modèles Cisco sur lesquels elle est supportée n'étant pas accessibles financièrement, l'expérimentation n'a pas pu aboutir.

Les recherches effectuées semblent toutefois indiquer que les possibilités de personnalisation sont complètes.

8.6 Translations d'adresses avec un NAT64/DNS64

Introduction

Deux expérimentations sont réalisées pour le NAT64/DNS64 : en mode *stateless* et en mode *stateful*. L'objectif est d'évaluer la faisabilité de chacun avec les équipements disponibles, et comparer les contraintes liées aux deux modes.

Un réseau uniquement IPv6 devra réussir à communiquer avec un réseau entièrement IPv4, sans utiliser de tunnels et ainsi s'affranchir de ses contraintes. Les machines dans le réseau de provenance devront faire résoudre leurs requêtes DNS par un serveur DNS64. Les faux `AAAA` générés par celui-ci devront utiliser le préfixe prévu par la norme.

8.6.1 Stateless

Cette expérimentation utilise une machine GNU/Linux en l'absence de matériel réseau compatible.

La topologie et le plan d'adressage sont décrits dans la figure 8.6 page 99.

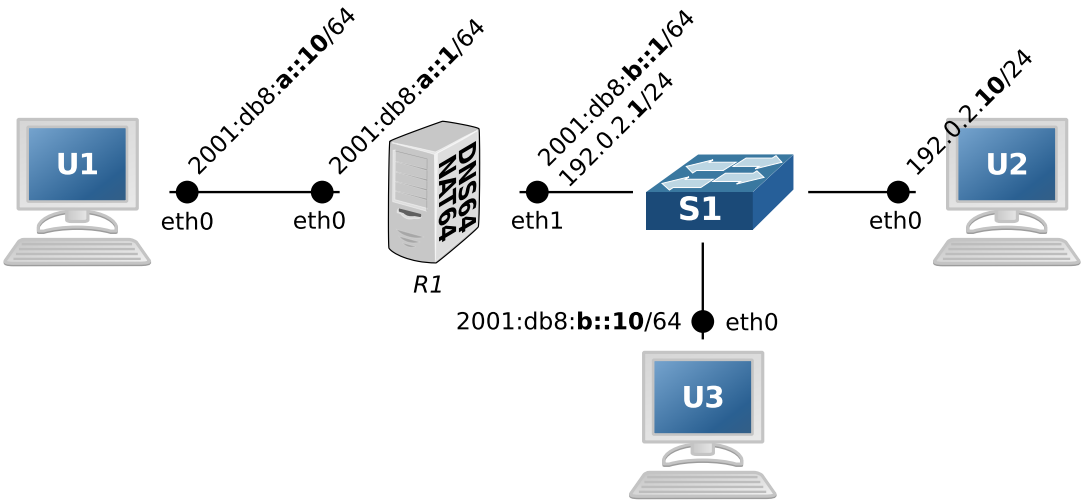


Figure 8.6 – *Installation liée à l'expérimentation du NAT64* stateless.

Le DNS64, le NAT64 et le serveur DNS ont été installés sur une même machine (dans un cas réel, au moins le serveur DNS serait externe), et le réseau Internet des schémas de la section 5.4.3 page 5.4.3 a été remplacé par un simple commutateur pour ne pas compliquer les configurations de tests.

Le serveur de passerelle (*R1*) utilise trois logiciels différents :

- **bind9 :** Répond aux requêtes DNS, en utilisant une base de données qui sera créée pour l'expérimentation.
- **totd :** Écoute sur le port 53 de la machine et sert de serveur DNS pour l'utilisateur U1. Il transmet toutes les requêtes DNS au vrai serveur DNS pour obtenir le `A` et/ou le `AAAA` correspondant à la question. S'il parvient à récupérer un `AAAA`, il transmet la réponse directement à U1, sinon il transforme le `A` en `AAAA` en injectant l'adresse IPv4 dans une adresse IPv6 du préfixe `64:ff9b::/96` avant de répondre. Bind peut directement assurer cette fonction s'il est à une version égale ou supérieure à la 9.8. Les deux versions seront expérimentées.
- **tayga :** Assure les fonctionnalités de NAT64 en créant une nouvelle interface qui servira de tunnel. Ce dernier se fait livrer tous les paquets adressés à une adresse IPv6 du préfixe `64:ff9b::/96`. Il se charge d'extraire l'IPv4 embarqué qu'il utilise comme adresse de destination du paquet IPv4 correspondant qu'il fait sortir à l'autre bout, et utilise une adresse privée de la plage `192.168.1.0/24` comme adresse source. Les paquets qui sortent de son tunnel sont transmis à l'interface `eth1` de sortie qui assure les fonctionnalités de NAT classiques. À l'inverse, il reçoit par l'autre extrémité tous les paquets IPv4 à destination d'une adresse de la plage `192.168.1.0/24` pour reproduire des paquets IPv6 selon les règles qu'il a enregistrées et les transmettre à `eth0`.

Installation des logiciels (l'utilisation de Totd dépend de la version du Bind) :

```
1 R1# apt-get install totd bind9 conntrack
2 R1# wget http://ubuntu.wikimedia.org/ubuntu/pool/universe/t/tayga/tayga_0.9.2-4
    _i386.deb
3 R1# dpkg -i tayga_0.9.2-4_i386.deb
```

Conntrack est ajouté pour *logguer* le NAT44 par la suite.

Configuration du fichier de configuration DNS pour les tests avec *bind9* (*/etc/bind/db.example*) :

```
$TTL 604800
@ IN SOA example. root.example. (
           3   ; Serial
      604800   ; Refresh
       86400   ; Retry
     2419200   ; Expire
      604800 ) ; Negative Cache TTL
;
@ IN NS example.
@ IN AAAA 2001:db8::1

u1 IN AAAA 2001:db8:a::10
u2 IN A 192.0.2.10
u3 IN AAAA 2001:db8:b::10
```

Création de la zone *example* (*/etc/bind/zone.example*) :

```
zone "example" {
  type master;
  file "/etc/bind/db.example";
};
```

Prise en compte de la nouvelle zone par Bind (ajouter à la fin de */etc/bind/named.conf*) :

```
include "/etc/bind/zones.example";
```

Si Totd est utilisé par la suite, Bind doit écouter sur un autre port que le 53 :

```
R1# sed 's/listen-on-v6/listen-on-v6 port 5300/' -i /etc/bind/named.conf.options
```

Et le NAT64 doit être configuré (*/etc/totd.conf*) :

```
forwarder ::1 port 5300
prefix 64:ff9b::
port 53
```

Commentaires par ligne :

1. Serveur à utiliser pour obtenir une réponse pour la requête reçue (ici c'est le Bind qui vient d'être installé).
2. Préfixe à utiliser pour créer les adresses IPv6 factices lorsqu'il n'y aura qu'un `A` de disponible au niveau du Bind (transformera les `A` en `AAAA` simplement en collant l'IPv4 correspondante à la fin du préfixe).
3. Port d'écoute, qui correspond au port standard des serveurs DNS.

Sinon (Bind $\geq$ 9.8), il est possible de configurer le NAT64 directement avec Bind (*/etc/bind/named.conf.options*) :

```
        dns64 64:ff9b::/96 {
                mapped { !10/8; !172.16/12; !192.168/16; any; };
```

```
3                exclude { 64:ff9b::/96; };
4        };
```

Contrairement au cas avec Totd, il n'y a pas de *forwarder* puisque Bind connait directement les réponses, et on exclut en plus la conversion des plages d'adresses privées. Si le `AAAA` trouvé semble être une adresse 64, le NAT64 décidera de la recalculer lui-même. Ces deux contraintes sont optionnelles.

Dans tous les cas, redémarrage de Bind (et Totd si besoin) puis test de la configuration :

```
1 R1# service totd restart
2 R1# service bind9 restart
3 R1# dig +short @::1 -p 5300 u2.example
4 192.0.2.10
```

Configuration du futur tunnel NAT64 (*/etc/tayga.conf*) :

```
1 tun-device nat64
2 ipv4-addr 192.168.1.1
3 prefix 64:ff9b::/96
4 dynamic-pool 192.168.1.0/24
5 data-dir /var/local/tayga
```

Commentaires par ligne :

1. Nom de l'interface qui correspondra au tunnel NAT64 (ici « *nat64* »).
2. Adresse que Tayga utilisera comme source pour envoyer ses propres messages d'erreur (le `man` indique qu'elle peut appartenir à la plage fournie ligne 4, et qu'elle sera transformée en IPv6 avec le préfixe utilisé pour le nat64 pour communiquer en IPv6).
3. Préfixe utilisé par le DNS64 pour créer les adresses IPv6 factices (RFC 6052).
4. Plage d'adresses IPv4 à disposition du NAT64 pour renseigner le champ IP source des paquets IPv4 qu'il sortira de son tunnel (et avec lesquelles il récupérera les réponses dans l'autre sens). Une plage privée a été choisie plutôt qu'une plage de documentation pour bien mettre en avant la nécessité du NAT par la suite.
5. Un fichier *dynamic.map* sera enregistré dans ce dossier, qui permettra de retrouver les associations IPv6-IPv4 en cas de redémarrage, pour les sessions en cours. Chaque adresse de ce fichier est réservée pour 2h04 au maximum après le dernier paquet traduit (pour un pool en /24, il y aura donc 254 adresses IPv6 différentes maximum sur une période de deux heures, la 255ième recevant un message ICMPv6 *unreachable*).

Création des adresses et des routes pour le tunnel (ainsi que du dossier pour le fichier de la table dynamique) :

```
1 R1# mkdir /var/local/tayga/
2 R1# tayga --mktun
3 R1# ip link set nat64 up
4 R1# ip route add 64:ff9b::/96 dev nat64
5 R1# ip route add 192.168.1.0/24 dev nat64
```

Commentaires par ligne :

1. Création de l'interface *nat64* (tunnel).
2. Activation de l'interface.
3. Tous les paquets IPv6 à destination d'une adresse IPv6 appartenant au préfixe utilisé par le DNS64 pour créer des adresses factices doivent passer par le tunnel pour que celui-ci puisse extraire les adresses IPv4 de destination et traduire les entêtes.
4. Dans le sens inverse, puisque les paquets sortent du tunnel en IPv4 avec une adresse source prise dans la plage définie dans sa configuration, il faut que cette même plage soit routée par défaut dans le tunnel pour que celui-ci puisse traduire les réponses IPv4 en IPv6.

En sortant du tunnel, les paquets IPv6 métamorphosés en paquets IPv4 utilisent des adresses IP source privées (`192.168.1.0/24`). Il faut donc que la machine assume également un rôle de NAT classique pour que les paquets sortant utilisent l'adresse publique (`192.0.2.1`) de l'interface de sortie (`eth1`) :

```
R1# iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
R1# iptables -A FORWARD -i eth1 -o nat64 -m state --state RELATED,ESTABLISHED -j
    ACCEPT
R1# iptables -A FORWARD -i nat64 -o eth1 -j ACCEPT
```

Enfin, la machine agissant comme routeur, il faut activer ses fonctions de routage au niveau du noyau :

```
R1# sysctl -w net.ipv4.conf.all.forwarding=1
R1# sysctl -w net.ipv6.conf.all.forwarding=1
```

Configuration IP de la machine, conformément au plan d'adressage (vérifier que le noyau a bien créé les routes correspondantes aux adresses des interfaces) :

```
R1# ip addr add 2001:db8:a::1/64 dev eth0
R1# ip addr add 2001:db8:b::1/64 dev eth1
R1# ip addr add 192.0.2.1/24 dev eth1
```

Lancement du démon de Tayga qui se chargera d'assurer les fonctions de NAT64 du tunnel (mode *debug*) :

```
R1# tayga -d
```

Configuration IP et DNS (interroge par défaut le DNS64 de sa passerelle) de la machine utilisateur U1 :

```
U1# ip addr add 2001:db8:a::10/64 dev eth0
U1# ip route add default via 2001:db8:a::1 dev eth0
U1# echo 'nameserver 2001:db8:a::1' > /etc/resolv.conf
```

Test du DNS64 depuis U1 avec la machine U2 (uniquement IPv4) :

```
U1# host u2.example
u2.example has address 192.0.2.10
u2.example has IPv6 address 64:ff9b::c000:20a
```

Configuration IP de la machine utilisateur U2 :

```
U2# ip addr add 192.0.2.10/24 dev eth0
U2# ip route add default via 192.0.2.1 dev eth0
```

Configuration IP de la machine utilisateur U3 :

```
U3# ip addr add 2001:db8:b::10/64 dev eth0
U3# ip route add default via 2001:db8:b::1 dev eth0
```

Test de U1 à U2 (vers une IPv4) :

```
U1# ping6 u2.example
PING u2.example(64:ff9b::c000:20a) 56 data bytes
64 bytes from 64:ff9b::c000:20a: icmp_seq=4 ttl=61 time=0.568 ms
64 bytes from 64:ff9b::c000:20a: icmp_seq=5 ttl=61 time=0.388 ms
64 bytes from 64:ff9b::c000:20a: icmp_seq=6 ttl=61 time=0.352 ms
```

Affichage de Tayga côté serveur :

```
assigned new pool address 192.168.1.2 (2001:db8:a::10)
```

Traces de Tshark sur les différentes interfaces de la passerelle durant le `ping` :

```
Capturing on eth0
  0.000000 2001:db8:a::10 -> 64:ff9b::c000:20a ICMPv6 Echo request
  0.000539 64:ff9b::c000:20a -> 2001:db8:a::10 ICMPv6 Echo reply

Capturing on nat64
  0.000000 2001:db8:a::10 -> 64:ff9b::c000:20a ICMPv6 Echo request
  0.000173 192.168.1.2 -> 192.0.2.10 ICMP Echo (ping) request
  0.002094  192.0.2.10 -> 192.168.1.2 ICMP Echo (ping) reply
  0.002114 64:ff9b::c000:20a -> 2001:db8:a::10 ICMPv6 Echo reply

Capturing on eth1
  0.000000   192.0.2.1 -> 192.0.2.10 ICMP Echo (ping) request
  0.000229  192.0.2.10 -> 192.0.2.1  ICMP Echo (ping) reply
```

Test de U1 à U3 (vers une IPv6, aucun paquet dans le tunnel) :

```
U1# ping6 u3.example
PING u3.example(2001:db8:b::10) 56 data bytes
64 bytes from 2001:db8:b::10: icmp_seq=1 ttl=63 time=0.390 ms
64 bytes from 2001:db8:b::10: icmp_seq=2 ttl=63 time=0.399 ms
64 bytes from 2001:db8:b::10: icmp_seq=3 ttl=63 time=0.387 ms
```

Tayga dispose d'un service avec un démon (les deux lignes du milieu permettent de déléguer au service la charge de créer les routes pour le préfixe IPv6 du DNS64 ainsi que la plage IPv4 du NAT64, puis de créer les règles netfilter pour le NAT44 en sortie) :

```
R1# sed 's/RUN="no"/RUN="yes"/' -i /etc/default/tayga
R1# sed 's/CONFIGURE_IFACE="no"/CONFIGURE_IFACE="yes"/' -i /etc/init.d/tayga
R1# sed 's/CONFIGURE_NAT44="no"/CONFIGURE_NAT44="yes"/' -i /etc/init.d/tayga
```

```
4 R1# service tayga start
```

Malheureusement, Tayga n'intègre aucune fonctionnalité concernant les journeaux systèmes, pourtant indispensable pour retrouver l'origine d'une connexion en cas de requête judiciaire. Puisque le fichier de la table dynamique est réécrit entièrement à chaque nouvelle association (et donc inexploitable), la seule solution trouvée est de récupérer les messages de sortie du mode *debug* (qui semble n'ajouter que cette fonctionnalité). Le service devrait donc être désactivé au démarrage.

Envoyer en temps réel les associations IPv6/IPv4 de Tayga à Syslog :

```
1 R1# (stdbuf -oL tayga -d | grep --line-buffered assigned | logger -t NAT64) &
```

Envoyer en temps réel les nouvelles connexions avec les associations IPv4 privée/publique à Syslog :

```
1 R1# (conntrack -E -o extended | logger -t NAT64) &
```

Aperçu d'une connexion de U1 vers U2 sur le port 80 (dans */var/log/user.log*, paramètre par défaut de logger) :

```
1 Jul 13 16:02:11 R1 NAT64: assigned new pool address 192.168.1.2 (2001:db8:a::10)
2 Jul 13 16:02:11 R1 NAT64:  [NEW] ipv4    2 tcp     6 120 SYN_SENT src=192.168.1.2
    dst=192.0.2.10 sport=49069 dport=80 [UNREPLIED] src=192.0.2.10 dst=192.0.2.1
    sport=80 dport=49069
```

8.6.2 Stateful

Le NAT64 *stateful* n'est actuellement disponible que sur les Cisco ASR 1000 et les CRS-1 [3].

Côté GNU/Linux, il existe le projet Ecdysis disponible aussi sous BSD. Malheureusement, la fondation qui soutient ce projet a pointé quelques gros problèmes [4].

Dans le cadre des tests effectués pour la conception de ce document, Cisco (par l'intermédiaire de l'intégrateur Axians) a fourni un équipement de pare-feu ASA 5505 avec une version du système ASA en version bêta, qui supporte la création de règles NAT64 *stateful*. C'est donc sur ce matériel, qui devrait être accessible pour tous prochainement, que cette expérimentation est réalisée.

La topologie et le plan d'adressage sont décrits dans la figure 8.7 page 105.

Le NAT *stateful* n'impose pas de passer par une plage d'adresse interne. Par contre, il faut bien différencier l'adresse d'interconnexion de l'adresse de sortie du NAT, qui ne peuvent pas être regroupées en une seule (le boîtier ADSL d'un particulier n'a qu'une IPv4 pour sortir, mais son FAI communique avec elle via un lien d'interconnexion qui utilise une adresse que l'abonné ne connait pas).

Informations supplémentaires :

- IPv4 publique de sortie du NAT : `192.0.2.2`
- Préfixe IPv6 utilisé pour transformer les `A` en `AAAA` : `2001:db8:c::/96`

3. Cf. Tableau 4 ici : http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6553/white_paper_c11-676278.html.
4. http://tnc2010.terena.org/files/jean_philippe_dionne.pdf

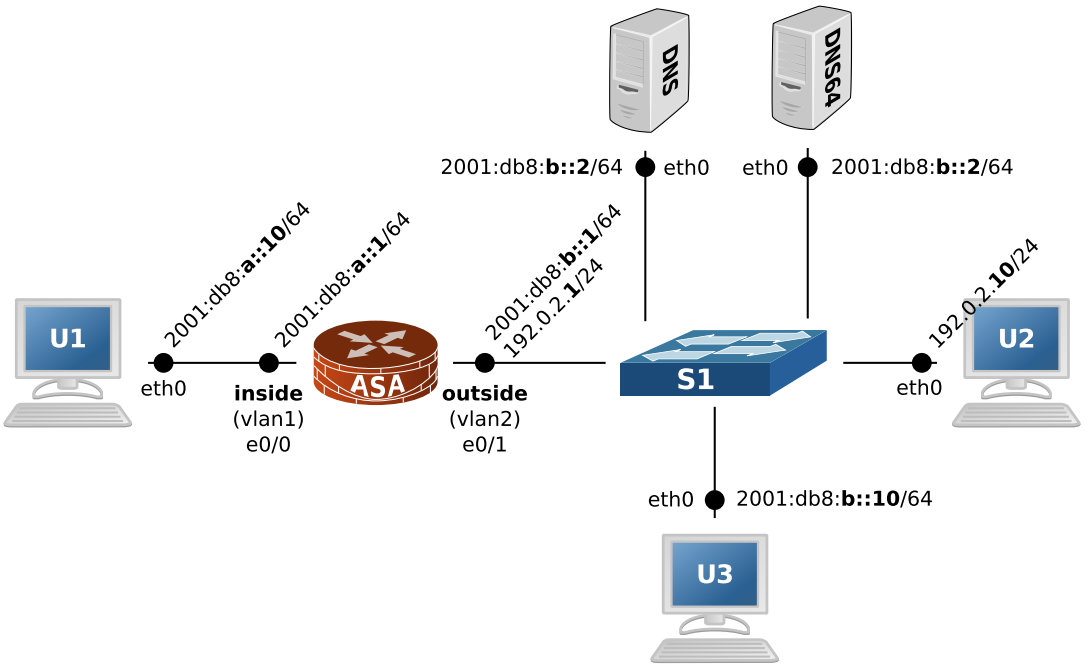


Figure 8.7 – *Installation liée à l'expérimentation du NAT64* stateful.

– Le serveur DNS n'assure pas lui-même les fonctionnalités de NAT64 dans cet exemple : cette situation est plus proche de ce qui sera probablement mis en place dans un premier temps, sur le réseau d'une institution. Le serveur DNS64 se contentera donc d'interroger les serveurs DNS habituels, qui ne seront pas modifiés le temps des tests. Il pourra utiliser Bind (`apt-get install bind9`) si sa version est égale ou supérieure à 9.8, sinon ce sera Totd (`apt-get install totd`).

Contrairement à l'expérimentation en mode *stateless*, le préfixe `64:ff9b::/96` destiné à être utilisé pour embarquer des adresses IPv4 ne peut pas être utilisé pour la traduction des `A` en `AAAA`. Il n'y a aucune raison logique à ça (ces adresses ne sortent pas du réseau) mais l'IOS refuse catégoriquement :

```
NAT64/46 real destination/source network cannot contain 0.0.0.0/0 when well known
    prefix used.
```

Configuration des interfaces de l'ASA :

```
ASA> en
ASA# conf t
ASA(config)# int vlan 1
ASA(config-if)# nameif inside
ASA(config-if)# security-level 1
ASA(config-if)# ipv6 address 2001:db8:a::1/64
ASA(config-if)# int vlan 2
ASA(config-if)# nameif outside
ASA(config-if)# security-level 2
ASA(config-if)# ipv6 address 2001:db8:b::1/64
ASA(config-if)# ip address 192.0.2.1/64
ASA(config-if)# int ethernet 0/0
ASA(config-if)# switchport access vlan 1
```

```
ASA(config-if)# no shut
ASA(config-if)# int ethernet 0/1
ASA(config-if)# switchport access vlan 2
ASA(config-if)# no shut
ASA(config-if)# exit
```

Configuration des objets sur l'ASA :

```
ASA(config)# object network ipv4-publique
ASA(config-network-object)# host 192.0.2.2
ASA(config-network-object)# object network ipv6-dns64
ASA(config-network-object)# subnet 2001:db8:c::/96
ASA(config-network-object)# object network ipv6-inside
ASA(config-network-object)# subnet 2001:db8:a::/64
ASA(config-network-object)# exit
```

Configuration du NAT64 sur l'ASA à partir des objets :

```
ASA(config)# nat (inside,outside) source static ipv6-inside ipv4-publique
   destination static ipv6-dns64 any
```

Version ASDM (*Configuration* > *Firewall* > *NAT Rules* > *Add*) en figure 8.8 page 106.

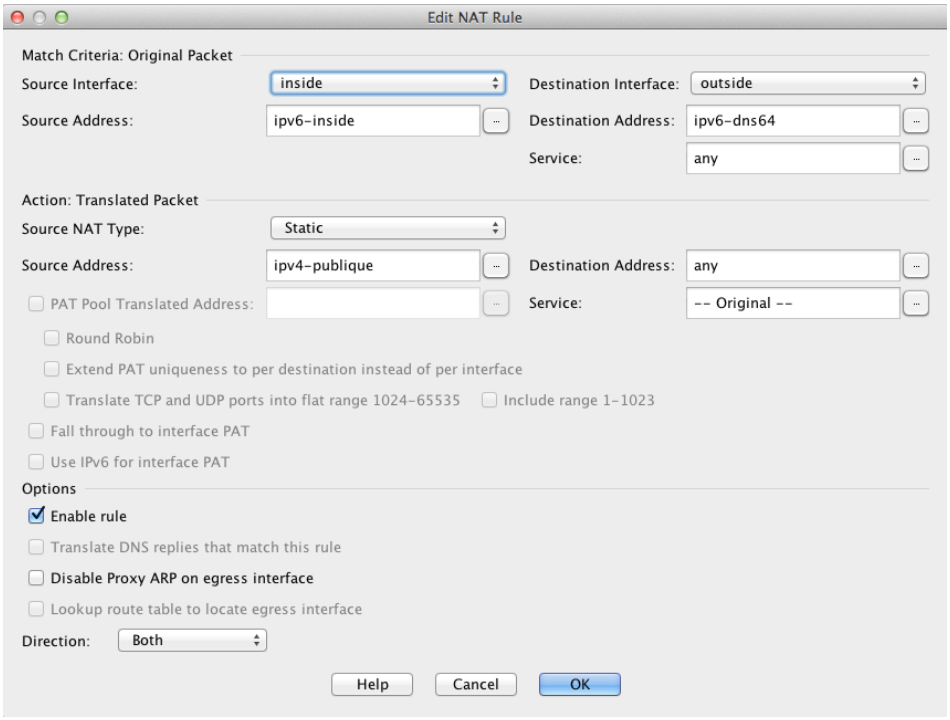


Figure 8.8 – *Configuration de la règle de NAT64* stateful *avec l'ADSM pour IOS 9.*

Configuration du pare-feu (aucun filtrage et journaux systèmes en mode *debug*) :

```
ASA(config)# access-list global_access extended permit ip any any log debugging
```

Version ASDM (*Configuration* > *Firewall* > *Access Rules*) en figure 8.9 page 107.

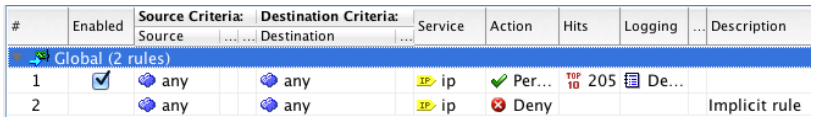


Figure 8.9 – *Autorisation de tous les trafics avec l'ADSM pour IOS 9.*

Configuration du serveur DNS :

```
DNS# ip addr add 2001:db8:b::2/64 dev eth0
DNS# // suite idem que pour l'infra stateless
```

Configuration du serveur DNS64 :

```
DNS# ip addr add 2001:db8:b::3/64 dev eth0
DNS# ip route add 2001:db8:a::/64 2001:db8:b::1 dev eth0
```

Si Totd est utilisé, configuration du DNS64 dans */etc/totd.conf* (cf. commentaires dans la version *stateless*) :

```
forwarder 2001:db8:b::2
prefix 2001:db8:c::
port 53
```

Sinon (Bind $\geq$ 9.8), configuration du DNS64 avec Bind (*/etc/bind/named.conf.options*) :

```
        forwarders {
                2001:db8:b::2;
        };

        dns64 2001:db8:c::/96 {
                recursive-only yes;
        };
```

Dans tous les cas, configuration du poste utilisateur U1 :

```
U1# ip addr add 2001:db8:a::10/64
U1# ip route add default via 2001:db8:a::1 dev eth0
U1# echo 'nameserver 2001:db8:b::3' > /etc/resolv.conf
```

Configuration de U2 et U3 : idem que pour l'expérimentation *stateless* (section 8.6.1 page 98).

Effectuer ensuite les mêmes tests de `ping` et de résolutions DNS que dans l'expérimentation *stateless*.

Une machine U1 entièrement IPv6 a été testée en conditions réelles, sur le réseau Lothaire, qui a lui-même nécessité quelques modifications supplémentaires :

1. Comme étudié dans la section consacrée à NDP (section 4.2 page 37), l'utilisation massive d'IP publiques nécessite du routage sur le réseau du FAI. En considérant que c'est l'adressage de cette expérimentation qui a été utilisée (ce qui n'est évidemment pas le cas), il faut rajouter une règle sur les routeurs en amont qui route le réseau `2001:db8:a::/64` vers `2001:db8:b::1` (la propagation de cette route a été concrétisée avec la directive *redistributes static* de IS-IS).

2. De la même façon, il faut annoncer des routes de `192.0.2.2/32` vers `192.0.2.1`.
3. Configurer les ACL du réseau pour autoriser les entrées/sorties vers ces réseaux, et autoriser les ports 53 pour les deux serveurs DNS.
4. Au niveau de l'ASA directement, il faut ajouter des routes par défaut pour l'IPv4 et l'IPv6 :

```
ASA(config)# ipv6 route outside ::/0 <ipv6-gateway>
ASA(config)# route outside 0.0.0.0 0.0.0.0 <ipv4-gateway>
```

5. Pour ne pas induire en erreur les autres machines du réseau attaché à la patte *outside*, il ne faut pas que celle-ci diffuse des annonces *Router Advertisement* :

```
ASA(config)# int vlan 2
ASA(config-if)# ipv6 nd suppress-ra
```

Concernant les journaux systèmes, l'ASA utilise son propre Syslog qui peut naturellement être configuré pour envoyer les données à un Syslog distant (exemple avec une connexion web de U1 vers U2) :

```
ASA(config)# logging enable
ASA(config)# logging buffered debugging
ASA(config)# do show logging
[...]
%ASA-6-302013: Built inbound TCP connection 3831 for inside:2001:db8:a::10/40524
   (192.0.2.2/40524) to outside:192.0.2.10/80 (2001:db8:c::c000:20a/80)
```

Une version ASDM est disponible via *Monitoring* > *Logging* > *Real-Time Log Viewer* > *View*.

Conclusion

Les deux expérimentations sont un succès, les objectifs ont été atteints, à l'exception de l'utilisation du préfixe normalisé avec l'ASA. Ce défaut est peut-être à mettre en lien avec l'aspect bêta du système, pour lequel une documentation truffée d'erreurs a aussi été fournie.

Le constat est toutefois relativement décevant au niveau de la compatibilité des matériels : la version *stateless* n'a pas pu être réalisée avec un équipement réseau Cisco, et la version *stateful* a nécessité de réclamer un prêt chez Cisco. L'IOS 9 semble faire un gros effort sur l'IPv6, et c'est à souhaiter que le NAT64 sera disponible sur un maximum d'équipement prochainement. Au niveau du DNS64, c'est une très bonne chose que Bind soit en capacité de le gérer nativement, mais la version installée n'est pas encore dans la Debian stable à ce jour.

La version *stateless* n'a pas l'air d'être à l'ordre du jour chez Cisco. L'expérimentation prouve que l'utilisation d'une plage interne est relativement contraignante : elle doit être bien dimensionnée, et ne pas correspondre à une plage qui sera potentiellement à router. L'absence de solution pour les jounaux systèmes pour Tayga est surprenante et la solution mise en œuvre manque de recul pour pouvoir évaluer sa capacité à suivre un trafic très important.

La conclusion générale de ce document prévoit en section 8.9 page 123, un retour d'expérience d'un mois avec l'expérimentation *stateful*.

8.7 Mobilité IPv6, DHCPv6 statique et relais DHCPv6

Introduction

La mobilité est une technologie qui est restée dans l'ombre pendant des années en IPv4, tant les contraintes liées aux NAT la portaient au rang de l'utopie. L'IPv6 prévoit une nouvelle version, et surtout de nouveaux espoirs en offrant un climat favorable à son utilisation à grande échelle.

L'expérimentation devra réussir à faire communiquer un poste utilisateur distant avec un poste utilisateur mobile. Ce dernier devra donc changer complètement de réseau, au cours d'une communication avec ce premier, sans que celle-ci ne souffre du changement. En mode sans optimisation de chemins, le réseau et la machine du côté du nœud distant ne doivent ni avoir connaissance de ce déplacement, ni avoir besoin de supporter la mobilité en particulier.

Pour être dans des conditions réelles de mobilités, des points d'accès wifi sont utilisés pour passer rapidement d'un réseau à l'autre. Pour respecter le plan d'adressage du schéma, et pour ajouter une autre notion à l'expérimentation, un serveur DHCPv6 est utilisé pour adresser automatiquement les machine en statique.

Expérimentation

La topologie et le plan d'adressage sont décrits dans la figure 8.10 page 109 (les noms des équipements respectent les conventions données dans le chapitre 7 page 81).

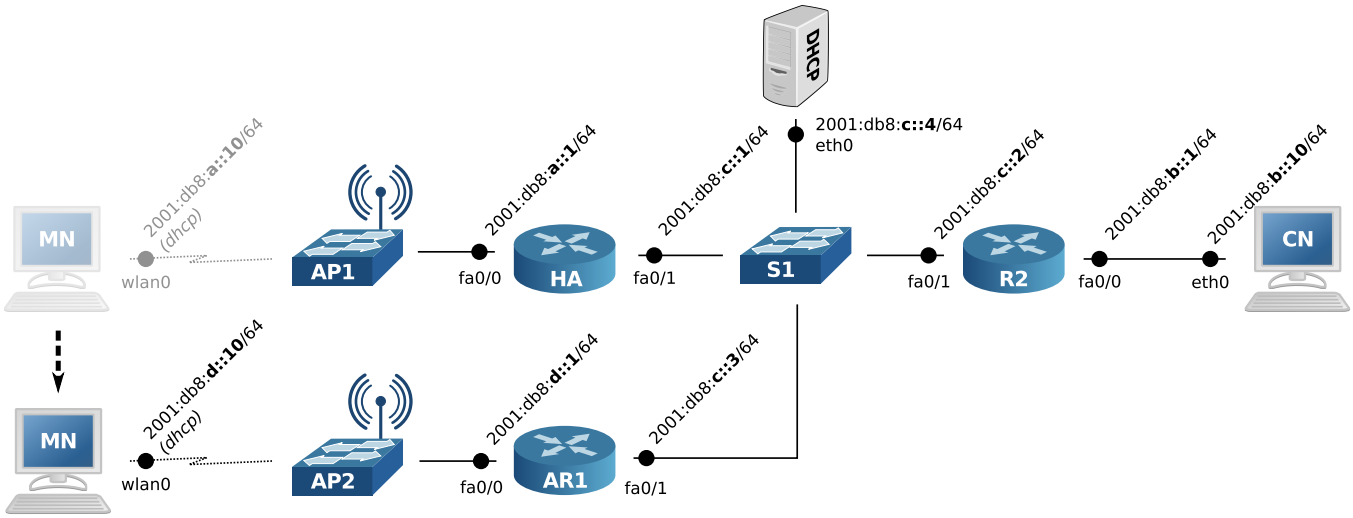


Figure 8.10 – *Installation liée à l'expérimentation de la mobilité IPv6.*

Configuration du point d'accès AP1 :

```
1 AP1> en
2 AP1# conf t
3 AP1(config)# dot11 ssid AP1
4 AP1(config)-dot11-ssid# authentification open
```

```
AP1(config-dot11-ssid)# guest-mode
AP1(config-dot11-ssid)# exit
AP1(config)# int Dot11Radio 0
AP1(config-if)# ssid AP1
AP1(config-if)# no shut
```

Configuration du point d'accès AP2 :

```
AP2> en
AP2# conf t
AP2(config)# dot11 ssid AP2
AP2(config)-dot11-ssid# authentification open
AP2(config-dot11-ssid)# guest-mode
AP2(config-dot11-ssid)# exit
AP2(config)# int Dot11Radio 0
AP2(config-if)# ssid AP2
AP2(config-if)# no shut
```

Configuration du routeur HA :

```
HA> en
HA# conf t
HA(config)# int fa 0/0
HA(config-if)# ipv6 addr 2001:db8:a::1/64
HA(config-if)# ipv6 dhcp relay destination 2001:db8:c::4
HA(config-if)# ipv6 mobile home-agent
HA(config-if)# no shut
HA(config-if)# int fa 0/1
HA(config-if)# ipv6 addr 2001:db8:a::1/64
HA(config-if)# no shut
HA(config-if)# exit
HA(config)# ipv6 unicast-routing
HA(config)# ipv6 route 2001:db8:b::/64 2001:db8:c::2
HA(config)# ipv6 route 2001:db8:d::/64 2001:db8:c::3
```

Configuration du routeur AR1 :

```
AR1> en
AR1# conf t
AR1(config)# int fa 0/0
AR1(config-if)# ipv6 addr 2001:db8:d::1/64
AR1(config-if)# no shut
AR1(config-if)# int fa 0/1
AR1(config-if)# ipv6 addr 2001:db8:c::3/64
AR1(config-if)# ipv6 dhcp relay destination 2001:db8:c::4
AR1(config-if)# no shut
AR1(config-if)# exit
AR1(config)# ipv6 unicast-routing
AR1(config)# ipv6 route 2001:db8:a::/64 2001:db8:c::1
AR1(config)# ipv6 route 2001:db8:b::/64 2001:db8:c::2
```

Configuration du routeur RCN :

```
RCN> en
RCN# conf t
RCN(config)# int fa 0/0
RCN(config-if)# ipv6 addr 2001:db8:b::1/64
```

```
RCN(config-if)# no shut
RCN(config-if)# int fa 0/1
RCN(config-if)# ipv6 addr 2001:db8:c::2/64
RCN(config-if)# no shut
RCN(config-if)# exit
RCN(config)# ipv6 unicast-routing
RCN(config)# ipv6 route 2001:db8:a::/64 2001:db8:c::1
RCN(config)# ipv6 route 2001:db8:d::/64 2001:db8:c::3
```

Configuration du serveur DHCP :

```
DHCP# ip addr add 2001:db8:c::4/64 dev eth0
DHCP# apt-get install isc-dhcp-server
```

Configuration du fichier */etc/dhcp/dhcpd6.conf* du serveur DHCP (adresse MAC de `wlan0` : `00:21:6a:27:9d:ea`) :

```
authoritative;

subnet6 2001:db8:c::/64 {}

subnet6 2001:db8:a::/64 {
        host AP1Client {
                host-identifier option dhcp6.client-id 00:03:00:01:00:21:6a:27:9d:
                    ea;
                fixed-address6 2001:db8:a::10;
        }
}

subnet6 2001:db8:d::/64 {
        host AP2Client {
                host-identifier option dhcp6.client-id 00:03:00:01:00:21:6a:27:9d:
                    ea;
                fixed-address6 2001:db8:d::10;
        }
}
```

Lancement du serveur DHCP :

```
DHCP# touch /var/lib/dhcp/dhcpd6.leases
DHCP# dhcpd -6 -f -cf /etc/dhcp/dhcpd6.conf
```

Configuration de la machine utilisateur CN :

```
CN# ip addr add 2001:db8:b::10/64 dev eth0
CN# ip route add default via 2001:db8:b::1 dev eth0
```

Configuration de la machine utilisateur MN (à noter que CentOS semble avoir un paquet *mipv6-daemon* dans ses dépôts) :

```
MN# ip addr add 2001:db8:a::10/64 dev wlan0
```

```
MN# ip route add default via 2001:db8:a::1 dev wlan0
MN# apt-get install git autoconf automake bison flex libssl-dev indent ipsec-
    tools
MN# git clone git://git.umip.org/umip.git
MN# cd umip
MN(umip)# autoreconf -i
MN(umip)# CPPFLAGS='-isystem /usr/src/linux/include/' ./configure --enable-vt
MN(umip)# make && make install
```

Configuration du fichier */usr/local/etc/mip6d.conf* de MN :

```
NodeConfig MN;
DebugLevel 10;

MnHomeLink "wlan0" {
  HomeAgentAddress 2001:db8:a::1;
  HomeAddress 2001:db8:a::10/64;
}

UseMnHaIPsec disabled;
```

Pour changer de point d'accès en ligne de commande, récupérer la MAC des AP ainsi que le canal utilisé :

```
MN# iwlist wlan0 scan | grep AP -B 5
          Cell 01 - Address: 00:3A:98:C1:28:50
                    Channel:6
                    Frequency:2.437 GHz (Channel 6)
                    Quality=70/70 Signal level=-16 dBm
                    Encryption key:off
                    ESSID:"AP2"
--
          Cell 02 - Address: 00:1A:30:32:3E:00
                    Channel:3
                    Frequency:2.422 GHz (Channel 3)
                    Quality=70/70 Signal level=-21 dBm
                    Encryption key:off
                    ESSID:"AP1"
```

Se connecter à AP1 puis lancer le démon MIPL (le driver de la carte utilisée nécessite d'éteindre l'interface) :

```
MN# ifconfig wlan0 down
MN# iwconfig wlan0 essid AP1 channel 3 ap '00:1A:30:32:3E:00'
MN# dhclient -6 -D LL wlan0
MN# mipd6
```

Vérification du fonctionnement du DHCP et du démon (le tunnel ne s'est pas approprié la *home-address*) :

```
MN# ip -6 a
3: wlan0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 2001:db8:a::10/64 scope global home nodad
       valid_lft forever preferred_lft forever
    inet6 fe80::221:6aff:fe27:9dea/64 scope link
       valid_lft forever preferred_lft forever
17: ip6tnl1: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1460
    inet6 fe80::221:6aff:fe27:9dea/64 scope link
       valid_lft forever preferred_lft forever
```

Trace du démon MIPL en relation :

```
Tue Jul 24 13:13:55 md_update_router_stats: Adding CoA 2001:db8:a:0:221:6aff:fe27
   :9dea on interface (3)
Tue Jul 24 13:13:55 __md_discover_router: discover link on iface wlan0 (3)
Tue Jul 24 13:13:57 mn_addr_do_dad: DAD succeeded!
Tue Jul 24 13:13:57 mn_addr_do_dad: address = 2001:db8:a:0:0:0:0:10
Tue Jul 24 13:13:57 mn_move: 1775
Tue Jul 24 13:13:57 mn_move: in home net
== NON_MIP_CN_ENTRY ==
Home address   2001:db8:a:0:0:0:0:10
Care-of address 2001:db8:a:0:0:0:0:10
CN address     2001:db8:a:0:0:0:0:1
```

Passage à l'AP2 :

```
MN# ifconfig wlan0 down
MN# iwconfig wlan0 essid AP2 channel 6 ap '00:3A:98:C1:28:50'
MN# dhclient -6 -D LL wlan0
```

Trace du démon MIPL en relation :

```
Tue Jul 24 13:38:40 md_update_router_stats: Adding CoA 2001:db8:d:0:221:6aff:fe27
   :9dea on interface (3)
Tue Jul 24 13:38:40 mn_move: 1775
Tue Jul 24 13:38:40 mn_move: in foreign net
Tue Jul 24 13:38:40 mn_block_rule_add: blackhole is already set.
Tue Jul 24 13:38:40 mn_send_home_bu: 792
Tue Jul 24 13:38:40 mn_get_home_lifetime: CoA lifetime 2591995 s, HoA lifetime
   2590714 s, BU lifetime 262140 s
Tue Jul 24 13:38:40 mn_ro_pol_add: Adding default RO triggering policies for all
   Correspondent Nodes
Tue Jul 24 13:38:40 process_first_home_bu: New bule for HA
Tue Jul 24 13:38:40 bul_add: Adding bule
== BUL_ENTRY ==
Home address   2001:db8:a:0:0:0:0:10
Care-of address 2001:db8:d:0:0:0:0:10
CN address     2001:db8:a:0:0:0:0:1
```

```
Tue Jul 24 13:38:40 mh_send: sending MH type 5
from 2001:db8:a:0:0:0:0:10
to 2001:db8:a:0:0:0:0:1
```

Trace Wireshark concernant l'envoi du *Binding Update* de MN pour prévenir HA de son déplacement :

```
MN# tshark -i wlan0 -VR mipv6
Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
[...]
    Source: 2001:db8:d::10 (2001:db8:d::10)
    Destination: 2001:db8:a::1 (2001:db8:a::1)
[...]
Mobile IPv6 / Network Mobility
    Payload protocol: IGMP (0x02)
    Header length: 33 (272 bytes)
    Mobility Header Type: Unknown (106)
    Reserved: 0xff
    Checksum: 0xfe27
    Unknown MH Type

Frame 2: 78 bytes on wire (624 bits), 78 bytes captured (624 bits) on interface 0
[...]
    Source: 2001:db8:a::1 (2001:db8:a::1)
    Destination: 2001:db8:d::10 (2001:db8:d::10)
[...]
Mobile IPv6 / Network Mobility
    Payload protocol: IPv6 no next header (0x3b)
    Header length: 2 (24 bytes)
    Mobility Header Type: Binding Error (7)
    Reserved: 0x00
    Checksum: 0x29ce
    Binding Error
        Status: Unknown binding for Home Address destination option (1)
        Home Address: 2001:db8:a::10 (2001:db8:a::10)
```

Détection de l'erreur par MIPL, qui résume bien la situation (*BE* = *Binding Error*) :

```
Tue Jul 24 13:38:40 mn_recv_be: Got BE from HA, it does not understand us ?
```

Conclusion

L'absence de RFC publiées sur ce sujet (la RFC 3775 est en *PROPOSED STANDARD*) explique probablement pourquoi un mobile GNU/Linux en vient à utiliser un numéro de protocole qu'un *home agent* Cisco ne connait même pas. Outre le fait de rappeler l'importance des standards, le logiciel utilisé par le mobile semble parfois avoir d'autres imperfections (figure 8.11 page 116).

Même en s'acharnant, les possibilités de mobilité avec l'IPv6 ne semblent pas encore être pour

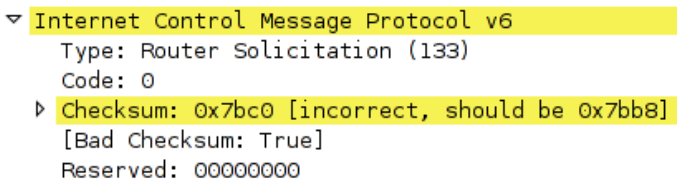


Figure 8.11 – *Erreur de* checksum *lors de l'expérimentation de MIPv6.*

aujourd'hui. Malgré des traces de projets très actifs entre 2004 et 2007, l'engouement semble retombée, et peu de systèmes ont suivi pour l'instant.

Concernant le DHCP statique, l'objectif est atteint puisque U1 a reçu des adresses précises selon le réseau sur lequel il s'est connecté en wifi. Après avoir passé beaucoup de temps à tenter de faire du statique avec le routeur Cisco, la conclusion semble être qu'ils ne supportent pas encore cette façon de faire. Le DHCP de ISC a très bien rempli son travail, mais la documentation laisse un peu à désirer côté IPv6, avec un `man` qui fournit la version IPv6 des paramètres quand ça lui chante, et sans vraie mise en parallèle. Les exemples sont restés exclusivement en IPv4, et le paquet Debian ne dispose pas encore de fichier de configuration d'exemple pour l'IPv6 (ce qui semble être le cas pour le paquet CentOS).

La fonction relais des routeurs Cisco ne pose aucun problème et se paramètre aisément.

8.8 Ajout dynamique d'adresse dans le DNS (DDNS) et pool DHCPv6

Introduction

L'absence de NAT sur les réseaux IPv6 impose de diffuser un nombre important d'adresses différentes sur Internet. Beaucoup de services nécessitant d'avoir un domaine et un reverse associés pour ne pas être considéré comme un *spammeur*, les attributions ne peuvent plus être réalisées à la main. L'injection d'entrées DNS via le DHCPv6 (autoconfiguration *stateful*) est une solution qui existait déjà en IPv4, avec le *Dynamic DNS*.

L'objectif de cette expérimentation est de tester l'avancement du DHCP de ISC et de Bind, pour assurer ces fonctions pour l'IPv6. Un poste utilisateur devra donc se faire attribuer une IP piochée aléatoirement dans une plage (*pool*) d'adresse définie dans le DHCP, et être ensuite en mesure de consulter le nom DNS ainsi que le reverse qui ont été associés dynamiquement.

Expérimentation

La topologie et le plan d'adressage sont décrits dans la figure 8.12 page 117.

Les deux IPv4 en bleu ont dû être ajoutées, contraint et forcé de constater que la version ISC du serveur DHCP (version 4.2.2 incluse) ne permet pas de renseigner une adresse IPv6 pour le serveur DNS à mettre à jour. Cette évolution arrivera probablement rapidement et est semble issue du même

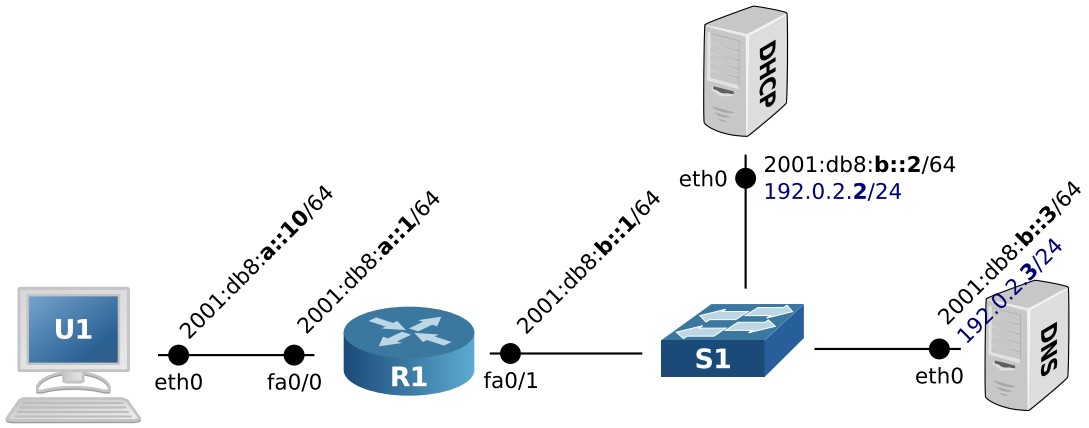


Figure 8.12 – *Installation liée à l'expérimentation du DDNS entre un serveur DHCP et un serveur DNS.*

état d'esprit que celui qui a conduit à attendre aussi longtemps pour mettre en œuvre le RDNSS, en considérant que les serveurs DNS seraient encore longtemps en IPv4. Dans ce cas de figure, il n'impose que la couche v4 sur les deux serveurs (plus d'explications plus bas).

Installation du serveur DNS et configuration IP :

```
DNS# apt-get install bind
DNS# ip addr add 2001:db8:b::3/64 dev eth0
DNS# ip route add 2001:db8:a::/64 2001:db8:b::1 dev eth0
DNS# ip addr add 192.0.2.3/24 dev eth0
```

Création d'une clé symétrique pour autoriser le DHCP à injecter des entrées dynamiquement :

```
DNS# cat /tmp/$(dnssec-keygen -a hmac-md5 -b 128 -n USER -K /tmp dhcpupdate).key
    | cut -d' ' -f7
HErio7xWBgGWPL8OFVcu0w==
```

Création d'un fichier de clé sur le serveur DNS (*/etc/bind/update.key*) :

```
key "dhcpupdate" {
  algorithm hmac-md5;
  secret "HErio7xWBgGWPL8OFVcu0w==";
};
```

Précisions :

- Le fichier */etc/bind/rndc.key* automatiquement généré sur certaines distributions pourrait être directement utilisé.
- Faire attention aux guillemets (qui ne devront pas toujours être présent, dans la suite).
- Vérifier que l'utilisateur de Bind (*bind* ou *named*) a bien le droit d'ajouter des fichiers dans */etc/bind/*.

Configuration du champ `AAAA` de U1 sur le serveur DNS (*/etc/bind/db.example*) :

```
$TTL  604800
@ IN  SOA example. root.example. (
           3   ; Serial
      604800   ; Refresh
       86400   ; Retry
     2419200   ; Expire
      604800 ) ; Negative Cache TTL
;
@ IN  NS  example.
@ IN  AAAA  2001:db8:b::3

u1  IN  AAAA  2001:db8:a::10
```

Configuration de son équivalent `PTR` (*/etc/bind/db.a.db8.2001*) :

```
$TTL 604800
a.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa IN SOA example. root.example. (
           3   ; Serial
      604800   ; Refresh
       86400   ; Retry
     2419200   ; Expire
      604800 ) ; Negative Cache TTL
;
@       IN      NS      example.
@       IN      AAAA    2001:db8:b::3

0.1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.a.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. PTR  u1
    .example.
```

Création des zones correspondantes, avec la clé qui autorise leur mise à jour (*/etc/bind/zones.example*) :

```
zone "example" {
        type master;
        file "/etc/bind/db.example";
        allow-update { key dhcpupdate; };
};

zone "a.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa" {
        type master;
        file "/etc/bind/db.a.db8.2001";
        allow-update { key dhcpupdate; };
};
```

Enfin, ajouter à la fin de */etc/bind/named.conf* :

```
include "/etc/bind/update.key";
include "/etc/bind/zones.example";
```

Redémarrer puis tester le serveur DNS :

```
DNS# service bind9 restart
DNS# dig +short AAAA @::1 u1.example
2001:db8:a::10
DNS# dig +short @::1 -x 2001:db8:a::10
u1.example.
```

Tester l'ajout dynamique d'entrées dans le DNS (**à réitérer** ensuite depuis la machine du serveur DHCP) :

```
DNS# nsupdate
> server ::1
> key dhcpupdate HErio7xWBgGWPL8OFVcu0w==
> zone example
> update add u2.example. 600 IN AAAA 2001:db8:a::20
> send
> quit
DNS# dig +short AAAA @::1 u2.example
2001:db8:a::20
```

Installation du serveur DHCP et configuration IP :

```
DHCP# apt-get install isc-dhcp-server
DHCP# ip addr add 2001:db8:b::2/64 dev eth0
DHCP# ip route add 2001:db8:a::/64 2001:db8:b::1/64 dev eth0
```

Configuration du DHCP (*/etc/dhcp/dhcpd6.conf*) :

```
authoritative;

ddns-updates on;
allow client-updates;
ddns-update-style interim;
do-forward-updates true;

ddns-domainname "example";
option dhcp6.name-servers 2001:db8:b::3;

key dhcpupdate {
  algorithm hmac-md5;
  secret HErio7xWBgGWPL8OFVcu0w==;
};

zone example. {
  key dhcpupdate;
  primary 192.0.2.3;
}

zone a.0.0.0.8.b.d.0.1.0.0.2.ip6.arpa. {
  key dhcpupdate;
```

```
23   primary 192.0.2.3;
24 }
25
26 subnet6 2001:db8:b::/64 {}
27
28 subnet6 2001:db8:a::/64 {
29   range6 2001:db8:a::1 2001:db8:a::ffff:ffff;
30   ddns-hostname = concat("dyn-", binary-to-ascii(16, 16, "-", substring(option
        dhcp6.ia-na, 16, 16)));
31 }
```

L'obligation d'utiliser des adresses IPv4 pour communiquer avec le serveur DNS provient du champ *primary* pour lequel le *parser* de Bind rejette les IPv6 :

```
1 /etc/dhcp/dhcpd6.conf line 26: expecting semicolon.
2   primary 2001:
3               ^
```

Le serveur accepte de se lancer en utilisant une adresse DNS (*ns.example*) plutôt qu'une adresse IPv6, mais ses habitudes de serveur préhistorique ne lui permettent pas d'arriver à ses fins lorsqu'il doit ajouter une entrée DNS dynamique :

```
1 ns.example: no A record associated with address.
```

Lancer le serveur en mode *debug* pour voir la trace :

```
1 DHCP# dhcpd -6 -f -d -cf /etc/dhcp/dhcpd6.conf
```

Configuration de R1 qui se contente de router et de faire relais DHCP :

```
1  R1> en
2  R1(config)# conf t
3  R1(config)# ipv6 unicast-routing
4  R1(config)# int f 0/0
5  R1(config-if)# ipv6 addr 2001:db8:a::1/64
6  R1(config-if)# ipv6 dhcp relay destination 2001:db8:b::2
7  Routeur(config-if)# ipv6 nd other-config-flag
8  Routeur(config-if)# ipv6 nd managed-config-flag
9  R1(config-if)# no shut
10 R1(config)# int f 0/1
11 R1(config-if)# ipv6 addr 2001:db8:b::1/64
12 R1(config-if)# no shut
```

Le routeur intervient dans cette expérimentation parce que U1 correspond à un Mac version Lion, et que celui-ci n'a pas d'option DHCP explicite pour l'IPv6. Il faut donc le configurer en *Automatique*, et forcer l'utilisastion du DHCPv6 via les drapeaux des *Router Advertisements*.

Pour tester la demande d'IPv6 en DHCP, brancher et débrancher la prise ethernet du Mac, ou la demander explicitement si c'est un GNU/Linux :

```
1 U1# dhclient -6 eth0
```

Trace du serveur DHCP :

```
Relay-forward message from 2001:db8:b::1 port 547, link address 2001:db8:a::1,
   peer address fe80::62fb:42ff:feef:e11a
Picking pool address 2001:db8:a::82d1:bea4
Sending Relay-reply to 2001:db8:b::1 port 547
Relay-forward message from 2001:db8:b::1 port 547, link address 2001:db8:a::1,
   peer address fe80::62fb:42ff:feef:e11a
Sending Relay-reply to 2001:db8:b::1 port 547
Added new forward map from dyn-2001-db8-a-0-0-0-82d1-bea4.example to 2001:db8:a
   ::82d1:bea4
Added reverse map from 4.a.e.b.1.d.2.8.0.0.0.0.0.0.0.0.0.0.0.0.0.0.a.0.0.0.8.b.d
   .0.1.0.0.2.ip6.arpa. to dyn-2001-db8-a-0-0-0-82d1-bea4.example
```

Trace au niveau de la machine du serveur DNS :

```
DNS# tshark -i eth0 -R 'dns.flags == 0x2800'
Running as user "root" and group "root". This could be dangerous.
Capturing on eth0
  9.847464   192.0.2.2 -> 192.0.2.3 DNS 265 Dynamic update 0xd220 SOA example
  9.859034   192.0.2.2 -> 192.0.2.3 DNS 276 Dynamic update 0x246b SOA a.0.0.0.8.b.
     d.0.1.0.0.2.ip6.arpa
```

Vérifier que U1 a bien récupéré l'adresse 2001:db8:a::82d1:bea4 et tester la nouvelle entrée du DNS :

```
U1# dig +short @2001:db8:b::3 -x 2001:db8:a::82d1:bea4
dyn-2001-db8-a-0-0-0-82d1-bea4.example
U1# dig +short AAAA @2001:db8:b::3 dyn-2001-db8-a-0-0-0-82d1-bea4.example
2001:db8:a::82d1:bea4
```

Conclusion

Le DDNS semble parfaitement fonctionner, et permettre une politique de confection des noms DNS dynamiquement qui n'est limitée que par l'imagination de l'administrateur (reproduire le comportement de l'autoconfiguration *stateless* en utilisant les MAC pourrait être intéressant). Malheureusement, la nécessité du lien IPv4 rappelle que le DNS IPv6 a été considéré trop longtemps non-prioritaire, considérant que la double couche sera toujours disponible.

Concernant le DHCPv6 dynamique, le fonctionnement est similaire à l'IPv4, mais le DHCP de ISC souffre d'un manque de documentation pour IPv6.

Conclusion

« *Any fool can use a computer. Many do.* » - Ted Nelson

8.9 Est-il possible de migrer son réseau interne en IPv6 uniquement, tout en continuant à bénéficier des services restés en IPv4 ?

Grâce à la solution du NAT64/DNS64, **la réponse est oui**.

Cette solution a été testée durant un mois sur le poste d'un utilisateur :

- La couche IPv4 a été désactivée (i.e. aucune interface ne disposait d'une IPv4).
- L'ASA qui a servi pour l'expérimentation du NAT64 *stateful* a été placé en aval de sa machine.
- Aucune configuration réseau n'a été effectuée sur la machine (en autoconfiguration et en distribuant l'adresse du DNS faisant office de DNS64).

Conformément au fonctionnement du NAT64, tous les sites web et services étaient accessibles en IPv6 pour la machine, qu'ils le supportent nativement ou non.

L'utilisateur a rencontré trois problèmes durant cette période :

1. Un client IRC qui supportait l'IPv6 mais qui s'acharnait à se connecter de préférence en IPv4, malgré l'absence d'interface compatible [5].
2. Un intranet d'école inaccessible à cause d'un DNS mal configuré, avec un faux `AAAA` disponible [6].
3. Un autre site web qui a posé le même problème [7].

De façon générale il n'aurait pu y avoir que quatre types de problème avec le NAT64 :

5. Un ticket a été posté à ce sujet, demandant de préférer automatiquement l'IPv6, comme c'est le cas sur la plupart des applications le supportant : http://bugs.irssi.org/index.php?do=details&task_id=842.
6. Les responsables techniques ont été prévenus (SI ESIAL-TELECOM Nancy). Solution temporaire au problème : `echo '2001:db8:c::193.50.40.1 www-intranet.esial.uhp-nancy.fr' >> /etc/hosts`.
7. Malgré une belle annonce : http://puppetlabs.com/blog/puppet-labs-helps-save-the-internet-ipv6-style/.

1. L'application a été mal écrite et utilise une IPv4 en dur, sans passer par le DNS (cas du client IRC).
2. Le DNS renseigne un faux `AAAA` (cas du site inaccessible).
3. L'application utilise explicitement le `A` du DNS s'il est disponible, en oubliant du même coup le `AAAA`.
4. L'application ne supporte pas l'IPv6, et ne prévoit donc pas d'interroger le `AAAA` du DNS.

Les trois premiers problèmes sont plutôt rares, et relèvent plus du *bug* que de la compatibilité. Le dernier pourrait être le plus courant, et concernera particulièrement les applications métier.

Un NAT64 n'ayant pas fondamentalement de raison d'être plus lent qu'un NAT44 classique (en oubliant la différence d'âge, qui peut jouer sur les performances) cette solution semble tout à fait envisageable dans un environnement de production. En prenant soin toutefois de vérifier la compatibilité des applications susceptibles d'être utilisées. Le réseau et les machines étant entièrement en IPv6, toute la puissance et la simplicité du nouveau protocole sont exploitables (échanges entre services Windows, téléphonie, etc.). Enfin, l'administration ne nécessite pas pour autant de double plan d'adressage ou de duplication des ACL.

Un NAT à peine plus particulier que ceux déjà mis en place subsiste, mais tendra à disparaître naturellement en étant de moins en moins sollicité au fil des années, sans qu'aucune intervention sur le réseau ou les postes clients ne soit nécessaire. Les nouveaux services mis en place sur le réseau seront conçus pour fonctionner avec l'IPv6, le rendant de plus en plus robuste et prêt pour l'avenir.

8.10 Est-il possible de se passer totalement de l'IPv4 ?

À cause de la maigre liste des sites web accessibles en IPv6 dans le top 100 des sites web les plus visités par les français (fournie en figure 1.2 page 10), **la réponse est non**.

Le mieux qui peut être fait actuellement est d'utiliser la solution du NAT64, et d'attendre patiemment qu'il ne soit plus sollicité. Ou d'utiliser une double pile sur les machines, si le réseau est suffisamment petit. Le jour où les sites pornographiques présents dans le top 100 seront accessibles en IPv6, ce sera un signal fort pour indiquer que le web est prêt à se passer totalement de l'IPv4.

8.11 Bilan des recherches et expérimentations

Le protocole apporte beaucoup de nouveautés très intéressantes et profite de l'expérience de l'IPv4 pour corriger ses principaux problèmes.

Beaucoup de solutions existent pour permettre à ceux qui n'ont pas de connexion IPv6 de naviguer au dessus de l'IPv4. Malheureusement, les efforts auraient plutôt dû être concentrés sur la disponibilités des services en IPv6, qui restent bien trop rares après tant d'années depuis les premières alertes de pénurie. Malgré tout, tous les systèmes d'exploitation récents supportent parfaitement l'IPv6, et de plus en plus de FAI proposent des plages à leurs abonnés.

L'autoconfiguration *stateless* (SLAAC) qui est l'une des nouveautés des plus remarquées souffre encore du retard de la normalisation du RDNSS qui n'est pas encore utilisable, faute de compatibilité suffisante avec les systèmes d'exploitation. Tant que les serveurs de nom de domaine devront être délivrés par DHCPv6, ce mode d'autoconfiguration n'aura pas un grand intérêt. Il pose également beaucoup de soucis au niveau de la sécurité (blocage de l'algorithme DAD, envois de *Router Advertisements* non-autorisés, etc.), et ne permet pas d'associer automatiquement un domaine et un reverse aux adresses. Le DHCPv6 aura donc probablement encore de beaux jours devant lui dans les réseaux d'entreprise.

Au delà des autres déceptions comme la mobilité, l'IPv6 fonctionne très bien et apporte beaucoup d'espoir en terme de simplicité des réseaux, tant avec l'abolition des NAT qu'avec ses adresses multicast et ses diverses optimisations.

Pour finir sur une note d'humour, voici à quoi pourrait ressembler la vie d'entreprise si le monde ne se décide pas à adopter IPv6 :

IPv6 - Are You Ready ? <http://www.youtube.com/watch?v=eYffYT2y-Iw>

A

Exemple de politique de sécurité

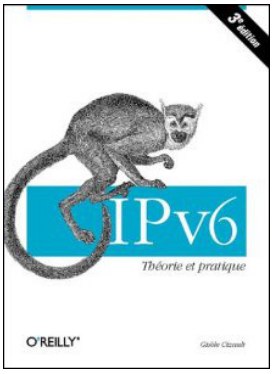
```
# INPUT #

# Maintenance of communication
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 1 -j ACCEPT # Destination Unreachable
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 2 -j ACCEPT # Packet too big
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 3 -j ACCEPT # Time Exceeded
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 4 -j ACCEPT # Parameter problem

# Connectivity checking
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 128 -j ACCEPT # Echo Request
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 129 -j ACCEPT # Echo Response

# Address configuration and router selection: allow in link-local only
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 133 -m hl --hl-eq 255 -j LOG --log-level warning --log-prefix "RS6 " # RS
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 133 -m hl --hl-eq 255 -j ACCEPT # RS
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 134 -m hl --hl-eq 255 -j ACCEPT # RA
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 135 -m hl --hl-eq 255 -j ACCEPT # NS
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 136 -m hl --hl-eq 255 -j LOG --log-level warning --log-prefix "NA6 " # NA
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 136 -m hl --hl-eq 255 -j ACCEPT # NA
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 141 -m hl --hl-eq 255 -j ACCEPT # Inverse NDS
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 142 -m hl --hl-eq 255 -j ACCEPT # Inverse NDA

# Link-local Multicast receiver: allow in link-local only
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 130 -m hl --hl-eq 255 -j ACCEPT # Listener Query
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 131 -m hl --hl-eq 255 -j ACCEPT # Listener Report
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 132 -m hl --hl-eq 255 -j ACCEPT # Listener Done
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 143 -m hl --hl-eq 255 -j ACCEPT # Listener Report v2

# SEND Certification Path Notification: allow in link-local traffic only
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 148 -m hl --hl-eq 255 -j ACCEPT # CPS
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 149 -m hl --hl-eq 255 -j ACCEPT # CPA

# Multicast Router messages: Advertisement, Solicitation, Termination
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 151 -m hl --hl-eq 255 -j ACCEPT # MRA
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 152 -m hl --hl-eq 255 -j ACCEPT # MRS
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 153 -m hl --hl-eq 255 -j ACCEPT # MRT

# Mobile IPv6
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 144 -j ACCEPT # Home Agent Address Discovery Request
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 145 -j ACCEPT # Home Agent Address Discovery Reply
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 146 -j ACCEPT # Mobile Prefix Solicitation
ip6tables -A ICMPv6_IN -p icmpv6 --icmpv6-type 147 -j ACCEPT # Mobile Prefix Advertisement
```

```
# Handling fragmentation
ip6tables -A ICMPv6_IN -m ipv6header --soft --header frag -j ACCEPT

# Applying ICMPv6_IN rules to general icmpv6 input
ip6tables -A INPUT -p icmpv6 -j ICMPv6_IN

# FORWARD #

# Maintenance of communication
ip6tables -A ICMPv6_FW -p icmpv6 --icmpv6-type 1 -j ACCEPT # Destination Unreachable
ip6tables -A ICMPv6_FW -p icmpv6 --icmpv6-type 2 -j ACCEPT # Packet too big
ip6tables -A ICMPv6_FW -p icmpv6 --icmpv6-type 3 -j ACCEPT # Time Exceeded
ip6tables -A ICMPv6_FW -p icmpv6 --icmpv6-type 4 -j ACCEPT # Parameter problem

# Connectivity checking, for now let the IPv6 world in
ip6tables -A ICMPv6_FW -p icmpv6 --icmpv6-type 128 -j ACCEPT # Echo Request
ip6tables -A ICMPv6_FW -p icmpv6 --icmpv6-type 129 -j ACCEPT # Echo Response

# Handling fragmentation
ip6tables -A ICMPv6_FW -m ipv6header --soft --header frag -j ACCEPT

# Applying rules
ip6tables -A FORWARD -p icmpv6 -j ICMPv6_FW
```

B

Références

« *The thing about quotes on the internet is that you cannot confirm their validity.* » - Abraham Lincoln



IPv6 : Théorie et pratique Écrit sous le nom d'emprunt Gisèle Cizault, il s'agit en réalité d'une œuvre du collectif d'universitaires et d'ingénieurs qui participe à la mise au point d'IPv6 au niveau international, au travers du G6 (association pour la promotion et le développement d'IPv6). Le livre est initialement édité par les éditions françaises de O'Reilly, qui ont disparues. Pour cette raison, et parce que le livre évoluait trop vite, il est disponible librement en version numérique [1] sur le site de l'association. Le faux nom Cizault vient du 6bone (littéralement *six os*), qui est une plate-forme d'expérimentation de l'IPv6. C'est probablement LA référence en matière d'IPv6, qui traite principalement de l'explication détaillée des protocoles, avec un peu de code C pour programmer des applications avec des *sockets* IPv6.

1. http://livre.g6.asso.fr

IPv6 : Principes et mise en œuvre Aux éditions ENI, il s'agit à ce jour du livre le plus récent (mai 2012) sur l'IPv6, disponible en français. Plutôt succinct sur l'aspect protocolaire, il s'attarde plutôt sur des explications concrètes des mécanismes liés à l'IPv6, avec des exemples de configuration. Il a aussi le mérite de s'intéresser de près aux aspects compatibilité des systèmes, tout en balayant un champ assez large des possibilités actuelles.

DNS and BIND on IPv6 Aux éditions O'Reilly, ce livre s'attarde sur toutes les possibilités liées à la résolution des noms de domaine en IPv6 ainsi qu'à la diffusion des informations du réseau via DHCPv6 (autoconfiguration *stateful*). Les premiers chapitres sont disponibles gratuitement sur Scribd [2] et SafariBooks [3].

Wiki ARIN Il s'agit d'une base de données communautaire [4] alimentée par un public professionnel qui partage ses notes qui concernent l'IPv6, notamment au travers des listes du NANOG (équivalent nord-américain du FRnOG).

World IPv6 Launch L'une des références parmi les acteurs importants de la démocratisation de l'IPv6, ce site [5] de l'Internet Society donne l'occasion au moins une fois par an de faire un point sur l'utilisation de l'IPv6 au travers du monde. Le RIPE propose aussi [6] des statistiques de ce type.

IPV6 HOWTO (GNU/Linux) Tiré de *The Linux Documentation Project*, ce tutoriel [7] très complet (bien que parfois un peu vieillissant) de Peter Bieringer, permet d'obtenir des solutions techniques précises pour l'utilisation d'IPv6 sur GNU/Linux.

Preparing an IPv6 Addressing Plan Proposé par le RIPE, ce document [8] d'une vingtaine de pages est destiné à assister la conception d'un plan d'adressage IPv6, à l'aide des *bonnes pratiques*.

IETF Cette référence pourrait faire partie de toutes les documentations informatiques, mais les RFC pour l'IPv6 sont une solution particulièrement efficace d'obtenir de manière sûre, à jour et complète, un descriptif détaillé des différents protocoles. Contrairement à ce qu'on pourrait craindre, elles sont très faciles à lire, et souvent plus claire que n'importe quel tutoriel déniché à coup de sueur depuis son moteur de recherche. Gagnez du temps, consultez directement les RFC et abonnez-vous au blog de Stéphane Bortzmeyer [9] qui propose une explication en français des nouveautés.

FAQ ISoc L'Internet Society met à disposition une FAQ [10] très intéressante sur les questions qui concernent l'IPv6.

Cheat Sheet Un format A4 recto-verso [11] avec tous les fondamentaux de l'IPv6, à afficher dans ses toilettes.

Quelques liens supplémentaires concernant la compatibilité des postes clients :

- Une page Wikipédia [12] fait le point sur la disponibilité de l'IPv6 sur les systèmes d'exploitation.

2. http://www.scribd.com/doc/63969331/DNS-and-BIND-on-IPv6
3. http://my.safaribooksonline.com/book/networking/dns/9781449308025
4. http://www.getipv6.info
5. http://www.worldipv6launch.org
6. http://v6day.ripe.net/cgi-bin/index.cgi
7. http://tldp.org/HOWTO/Linux+IPv6-HOWTO/index.html
8. http://www.ripe.net/lir-services/training/material/IPv6-for-LIRs-Training-Course/IPv6_addr_plan4.pdf/view
9. http://www.bortzmeyer.org
10. http://www.isoc.org/internet/issues/ipv6_faq.shtml
11. http://www.roesen.org/files/ipv6_cheat_sheet.pdf
12. https://en.wikipedia.org/wiki/Comparison_of_IPv6_support_in_operating_systems

- Un tableau [13] très intéressant qui récapitule les commandes systèmes fondamentales pour l'IPv6, pour différents systèmes d'exploitation.
- Analyse assez détaillée [14] (mais un peu ancienne) de la compatibilité des différentes technologies IPv6 sur les systèmes d'exploitation.

13. https://wikispaces.psu.edu/display/ipv6/IPv6+Rosetta+Stone
14. http://ipv6int.net/systems

C

Table des RFC

« *Let's say the docs present a simplified view of reality.* » [1] - Larry Wall (1990)

Ces différentes RFC [2] ont été citées dans ce document (certaines sont informationnelles) :

- **RFC 791 :** *INTERNET PROTOCOL*
- **RFC 1191 :** *Path MTU Discovery*
- **RFC 1338 :** *Supernetting : an Address Assignment and Aggregation Strategy*
- **RFC 1918 :** *Address Allocation for Private Internets*
- **RFC 2080 :** *RIPng for IPv6*
- **RFC 2081 :** *RIPng Protocol Applicability Statement*
- **RFC 2136 :** *Dynamic Updates in the Domain Name System (DNS UPDATE)*
- **RFC 2461 :** *Neighbor Discovery for IP Version 6 (IPv6)*
- **RFC 2464 :** *Transmission of IPv6 Packets over Ethernet Networks*
- **RFC 2473 :** *Generic Packet Tunneling in IPv6 Specification*
- **RFC 2606 :** *Reserved Top Level DNS Names*
- **RFC 2675 :** *IPv6 Jumbograms*
- **RFC 3007 :** *Secure Domain Name System (DNS) Dynamic Update*
- **RFC 3041 :** *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*
- **RFC 3053 :** *IPv6 Tunnel Broker*
- **RFC 3056 :** *Connection of IPv6 Domains via IPv4 Clouds*

1. http://groups.google.com/groups?selm=6940@jpl-devvax.JPL.NASA.GOV&hl=en
2. Pour consulter une RFC : http://tools.ietf.org/html/rfcXXXX.

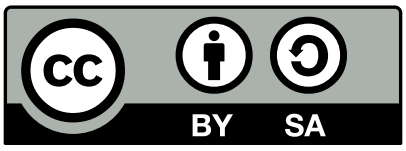
- **RFC 3068 :** *An Anycast Prefix for 6to4 Relay Routers*
- **RFC 3224 :** *Vendor Extensions for Service Location Protocol, Version 2*
- **RFC 3306 :** *Unicast-Prefix-based IPv6 Multicast Addresses*
- **RFC 3315 :** *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*
- **RFC 3484 :** *Default Address Selection for Internet Protocol version 6 (IPv6)*
- **RFC 3736 :** *Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6*
- **RFC 3775 :** *Mobility Support in IPv6*
- **RFC 3849 :** *IPv6 Address Prefix Reserved for Documentation*
- **RFC 3882 :** *Configuring BGP to Block Denial-of-Service Attacks*
- **RFC 3963 :** *Network Mobility (NEMO) Basic Support Protocol*
- **RFC 3972 :** *Cryptographically Generated Addresses (CGA)*
- **RFC 4140 :** *Hierarchical Mobile IPv6 Mobility Management (HMIPv6)*
- **RFC 4193 :** *Unique Local IPv6 Unicast Addresses*
- **RFC 4271 :** *A Border Gateway Protocol 4 (BGP-4)*
- **RFC 4260 :** *Mobile IPv6 Fast Handovers for 802.11 Networks*
- **RFC 4291 :** *IP Version 6 Addressing Architecture*
- **RFC 4380 :** *Teredo : Tunneling IPv6 over UDP through Network Address Translations (NATs)*
- **RFC 4389 :** *Neighbor Discovery Proxies (ND Proxy)*
- **RFC 4443 :** *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*
- **RFC 4472 :** *Operational Considerations and Issues with IPv6 DNS*
- **RFC 4861 :** *Neighbor Discovery for IP version 6 (IPv6)*
- **RFC 4862 :** *IPv6 Stateless Address Autoconfiguration*
- **RFC 4966 :** *Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status*
- **RFC 5214 :** *Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)*
- **RFC 5340 :** *OSPF for IPv6*
- **RFC 5342 :** *IANA Considerations and IETF Protocol Usage for IEEE 802 Parameters*
- **RFC 5568 :** *Mobile IPv6 Fast Handovers*
- **RFC 5635 :** *Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF)*
- **RFC 5737 :** *IPv4 Address Blocks Reserved for Documentation*
- **RFC 5969 :** *IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification*
- **RFC 6052 :** *IPv6 Addressing of IPv4/IPv6 Translators*
- **RFC 6106 :** *IPv6 Router Advertisement Options for DNS Configuration*
- **RFC 6146 :** *Stateful NAT64 : Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*
- **RFC 6275 :** *Mobility Support in IPv6*
- **RFC 6666 :** *A Discard Prefix for IPv6*

D

Figures et tableaux

Table des figures

Liste des tableaux



Les sources LaTeX et SVG sont disponibles sur http://ju.vg.

N'hésitez pas à le corriger, le compléter ou l'actualiser et à me contacter sur ✉ julien@vaubourg.com.