

Routeur Linux et IPv6 Free : une solution

La problématique :

Imaginons un Geek connecté à Internet via [Free](#). En temps que Geek il n'aura certainement pas activé la fonction routeur de sa freebox pour utiliser un bon vieux PC sous Linux pour faire le NAT et pouvoir rediriger des ports dans tous les sens. Derrière ce Linux se trouve, via une autre interface réseau un switch ou un hub sur les quels sont connecté quelques PCs.

Voilà-ti pas qu'un jour [Free propose de l'IPv6](#) ! Quelle bonne nouvelle !

On active l'option (gratuite) dans sa console de gestion, on reboot la freebox et nous voilà avec un /64 disponible sur le minuscule réseau entre la freebox et notre routeur Linux.

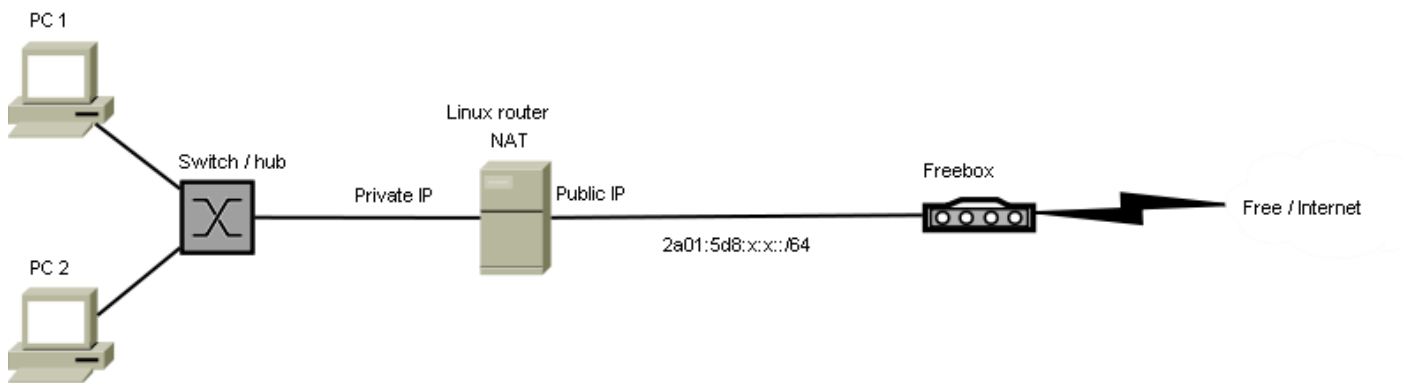


Fig. 1 - Configuration de base du Geek

Donc oui, notre routeur Linux va pouvoir avoir une adresse IPv6 sur son interface externe.

La belle affaire : et nos machines se trouvant derrière notre routeur Linux ?

En effet, le /64 fournit par Free est fait pour alimenter un segment ethernet et pas plus. Donc il n'est pas possible de router une partie de ce /64 vers un autre segment ethernet.

Une solution :

On fait comment alors ? NAT en IPv6 ? Beark ! L'IPv6 devrait justement permettre d'éviter ce genre de bidouille.

On pourrait également supprimer ce routeur Linux et donc laisser la freebox en mode routeur faire le NAT. Mais notre Geek ne pourrait plus jouer sur son routeur Linux.

La solution que je propose aujourd'hui est à mi chemin entre les deux solutions précédentes : conserver la fonction de NAT effectuée par le routeur Linux pour l'IPv4 et s'en passer pour l'IPv6.

On va donc créer un pont entre les deux interfaces de notre routeur Linux et le réserver au trafic IPv6. Ce pont sera donc un pont logiciel géré par la machine sous Linux. Ce pont se verra attribuer une simple règle permettant au trafic IPv6 d'être bridgé tout en laissant la gestion du reste du trafic au routeur Linux.

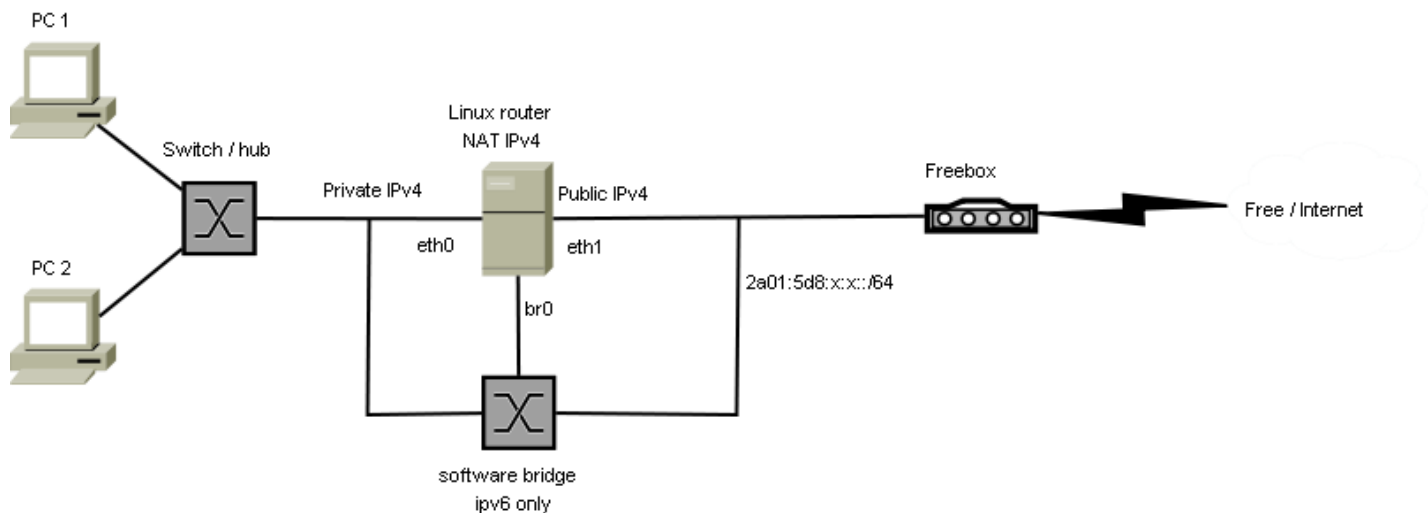


Fig. 2 - Un bridge logiciel uniquement pour l'IPv6

Commençons par créer ce pont virtuel :

```
brctl addbr br0
ifconfig br0 up
```

On intègre à ce pont les deux interfaces :

```
brctl addif br0 eth0
brctl addif br0 eth1
```

Ajoutons la règle ([brouting](#)) permettant d'utiliser le pont uniquement pour l'IPv6 :

```
ebtables -t broute -A BROUTING -p ! ipv6 -j DROP
```

En clair : tout ce qui n'est pas de l'IPv6 (donc IPv4, ARP, ...) est exclue du bridge et on fait comme si ça arrivait par l'interface normale (eth0 ou eth1 dans notre cas).

That's all.

Quelques précisions :

Si vous opérez sous Debian, vous aurez sans doute besoin des packages bridge-utils et ebtables.

Ici nous avons à exclure uniquement l'IPv6 car, contrairement à l'IPv4, nous n'utilisons pas de protocole comme arp (qui n'est pas de l'IP). Les neighbor sollicitations sont effectués en multicast.

Attention à ne pas laisser traîner d'éventuelles adresses IPv6 sur les interfaces du bridge. Pour que la machine Linux utilise l'IPv6 il faut configurer son interface br0.

L'utilisation d'un Neighbor Discovery Proxy tel que décrit dans la [RFC 4389](#) aurait certainement été plus propre, mais je n'ai trouvé aucune implémentation pour Linux.

Edit du 21/12/2007 : Il semblerait que le ndp proxy soit implémenté depuis le kernel [2.6.19](#) via le sysctl net.ipv6.conf.*.proxy_ndp. (Merci à kaouete pour l'info.)

Edit du 31/01/2008 : [Tutoriel sur l'utilisation du NDP sous Linux dans ce cas](#) (Merci à Patrick pour le lien). Je préfère le brouting. A vous de choisir.

Références :

[Linux bridging ebtables](#)

IPv6 2a01:e35:2f22:e3d0::2:2