R3.01 - IPv6

En raison de la disponibilité limitée des adresses Internet Protocol Version 4 (IPv4) codées sur 32 bits et offrant donc environ 4 millions d'adresses possibles et de l'expansion d'Internet parallèlement à la croissance de l'Internet of Things (IoT), l'Internet Protocol Version 6 (IPv6) commence à prendre forme comme un mal nécessaire. Les grands services modernes conçus pour creuser les lacunes d'IPv4, IPv6 dispose d'un espace d'adressage beaucoup plus grand compte tenu de sa structure de 128 bits, pouvant ainsi accueillir à la fois les connexions actuelles et futures car offrant 3,4 sextillions d'adresse possibles.

Néanmoins, le passage à IPv6 n'implique pas simplement un échange d'adresses. Il apporte des spécificités de routage supplémentaires qui modifient l'approche du mouvement des données dans les réseaux. Ces nouveaux paradigmes de routage associés à IPv6 offrent non seulement une meilleure utilisation des ressources, mais aussi des avantages supplémentaires dans des aspects tels que la sécurité, la communication mobile et la gestion des itinéraires.

Nous étudierons les défis techniques et de routage spécifiques propre à l'IPv6 et éluciderons les nouveaux modèles, les schémas d'adressage et les problèmes d'intégration avec la version quatre actuelle du protocole Internet…

SOMMAIRE

1 - MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL

A - Récupération de la VM

B - Création du LAB dans NETKIT

2 - CONFIGURATION ET VÉRIFICATION DES ADRESSES IPV4 ET IPV6

C - Révision IPv4

D - IPv6 : prise en main

3 - PARAMÉTRAGE ET ANALYSE DES PROPRIÉTÉS IPV6

E - IPv6 : paramétrage IP Statique de base

F - IPv6 : Adresse Multicast

G - IPv6 : Le protocole

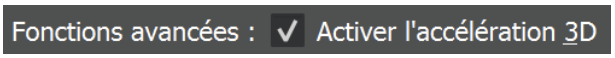
4 - ROUTAGE STATIQUE IPV6

H - Routage statique IPv6

1 - MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL

A - Récupération de la VM

Nous commençons par récupérer la Machine Virtuelle (VM) depuis le lien Google Drive puis la lançons en modifiant le paramètre d'affichage suivant :



Il est nécessaire d'activer l'accélération 3D car elle permet d'améliorer les performances graphiques et la gestion des ressources matérielles.

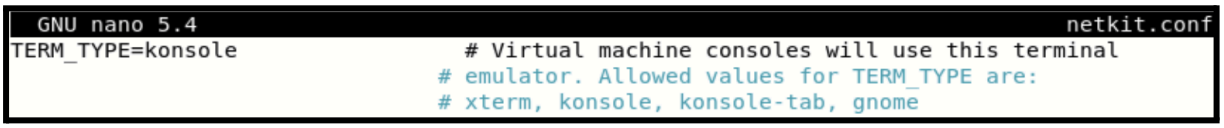
Et sachant que voici les différentes associations identifiant mot de passe :

Identifiant	**Mot de passe**
userrt	userrt
root	class

Nous pouvons donc débuter la création du LAB dans Netkit…

B - Création du LAB dans NETKIT

Avant toute autre modification, nous modifions le paramètre TERM_TYPE du fichier de configuration /TOOLS/netkit/netkit.conf et lui donnons la valeur "konsole" pour que les terminaux de nos VMs soient en mode console :



Une fois cela fait, nous allons dans un premier temps créer les différents équipements virtuels avec Netkit. Pour faire cela, nous allons ans le dossier /LAB-RT et modifions le fichier "lab.conf" au moyen de la commande *nano /LAB-RT/pc1.startup*, en adaptant les informations de cette manière :

```
 GNU nano 5.4                                                    /LAB-RT/lab.conf
LAB_DESCRIPTION="Lab IPv6 Netkit"
LAB_AUTHOR="ALAMELOU Rohan et BOUGHLEM Bilel"
pc1[0]=dca
r1[0]=dca
r1[1]=dcb
r2[0]=dcb
r2[1]=dcc
pc2[0]=dcc
```

Ce fichier configure la topologie réseau du LAB dans Netkit, associant chaque VM avec des interfaces et des liens, avec comme exemple, pc1[0]=dca signifiant que l'interface réseau eth0 de pc1 est reliée au segment dca. Et deux VM partageant le même segment sont directement interconnectées d'où PC1 et R1 directement interconnectés.

Nous pouvons à présent créer les fichiers contenant les configurations de base des équipements lorsqu'ils se lanceront, en commençant par PC1 dont nous créons le fichier de configuration au moyen de cette commande *nano /LAB-RT/pc1.startup*, puis ajoutons ceci :

```
 GNU nano 5.4                                                    pc1.startup
ip addr add 10.0.0.1/24 dev eth0
ip link set eth0 up
```

Puis R1 au moyen de la commande *nano /LAB-RT/r1.startup*, puis ajoutons ceci :

```
 GNU nano 5.4                                                    r1.startup
ip addr add 10.0.0.2/24 dev eth0
ip link set eth0 up
ip addr add 10.0.1.1/24 dev eth1
ip link set eth1 up
```

Puis R2 au moyen de la commande *nano /LAB-RT/r2.startup*, puis ajoutons ceci :

```
 GNU nano 5.4                                                    r2.startup
ip addr add 10.0.1.2/24 dev eth0
ip link set eth0 up
ip addr add 10.0.2.1/24 dev eth1
ip link set eth1 up
```

Puis PC2 au moyen de la commande *nano /LAB-RT/pc2.startup*, puis ajoutons ceci :

```
 GNU nano 5.4                                                    pc2.startup
ip addr add 10.0.2.2/24 dev eth0
ip link set eth0 up
```

Nous pouvons alors créer un dossier pour chaque VM au moyen de la commande suivante :

mkdir /LAB-RT/pc1 /LAB-RT/r1 /LAB-RT/r2 /LAB-RT/pc2

Et voici donc à quoi ressemble notre dossier /LAB-RT :

```
root@DebianIPv6:/LAB-RT# ls -l
total 872
-rw-r--r-- 1 root root         142  6 nov.  07:06 lab.conf
drwxr-xr-x 2 root root        4096  4 janv.  2023 pc1
-rw-r--r-- 1 root root 10740047872  4 janv.  2023 pc1.disk
-rw-r--r-- 1 root root          53  6 nov.  07:14 pc1.startup
drwxr-xr-x 2 root root        4096  6 nov.  07:28 pc2
-rw-r--r-- 1 root root          53  6 nov.  07:16 pc2.startup
drwxr-xr-x 2 root root        4096  6 nov.  07:28 r1
-rw-r--r-- 1 root root         106  6 nov.  07:22 r1.startup
drwxr-xr-x 2 root root        4096  6 nov.  07:28 r2
-rw-r--r-- 1 root root         106  6 nov.  07:24 r2.startup
```

Puis donnons les droits nécessaires au dossier /LAB-RT afin de pouvoir exécuter les script d'initialisation au moyen de la commande *chmod -R 755 /LAB-RT*, sachant que l'option *-R* permet d'appliquer les droits à tous les sous-répertoires et fichiers ce qui est le but et l'option 755 permet de donner les droits suivants : 7 pour le propriétaire (lire, écrire, exécuter), 5 pour le groupe (lire et exécuter, mais pas écrire) et 5 pour les autres (lire et exécuter uniquement).

Et voici donc à quoi ressemble le dossier une fois les nouveaux droits appliquées :

```
root@DebianIPv6:/LAB-RT# ls -l
total 872
-rwxr-xr-x 1 root root         142  6 nov.  07:06 lab.conf
drwxr-xr-x 2 root root        4096  4 janv.  2023 pc1
-rwxr-xr-x 1 root root 10740047872  4 janv.  2023 pc1.disk
-rwxr-xr-x 1 root root          53  6 nov.  07:14 pc1.startup
drwxr-xr-x 2 root root        4096  6 nov.  07:28 pc2
-rwxr-xr-x 1 root root          53  6 nov.  07:16 pc2.startup
drwxr-xr-x 2 root root        4096  6 nov.  07:28 r1
-rwxr-xr-x 1 root root         106  6 nov.  07:22 r1.startup
drwxr-xr-x 2 root root        4096  6 nov.  07:28 r2
-rwxr-xr-x 1 root root         106  6 nov.  07:24 r2.startup
```

Nous pouvons à présent tester nos configurations en démarrant les VMs en utilisant la commande *lstart* suivie du nom de la VM ou de l'option -f pour toutes les lancer :

```
--- Starting Netkit phase 2 init script ---

>>> Running pc1 specific startup script...
>>> End of pc1 specific startup script.


################################################

Lab directory (host): /LAB-RT
Version: <none>
Author:  ALAMELOU Rohan et BOUGHLEM Bilel
Email:   <none>
Web:     <none>
Description:
Lab IPv6 Netkit

################################################

--- Netkit phase 2 initialization terminated ---


pc1 login: root (automatic login)
Last login: Wed Nov  6 06:48:46 UTC 2024 on tty1
pc1:~#
```

Et tout fonctionne pour le moment…

2 - CONFIGURATION ET VÉRIFICATION DES ADRESSES IPV4 ET IPV6

C - Révision IPv4

Nous pouvons alors effectuer différents tests de connectivité en commençant par PC1 et R1 :

```
pc1:~# ping -c 4 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=3.05 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.309 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.255 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.211 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 0.211/0.957/3.056/1.212 ms
```

Puis R1 et R2 :

```
r1:~# ping -c 4 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=8.52 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=0.328 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=0.428 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=64 time=0.495 ms

--- 10.0.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3018ms
rtt min/avg/max/mdev = 0.328/2.444/8.528/3.513 ms
```

Puis R2 et PC2 :

```
r2:~# ping -c 4 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=64 time=6.72 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=64 time=0.246 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=64 time=0.182 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=64 time=0.464 ms

--- 10.0.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3018ms
rtt min/avg/max/mdev = 0.182/1.903/6.723/2.784 ms
```

Et enfin PC2 et PC1 :

```
pc2:~# ping -c 4 10.0.0.1
connect: Network is unreachable
```

Tous les tests fonctionnent sauf le dernier et c'est normal, puisqu'il manque une route entre les PC1 et PC2 pour les lier. Il aurait donc fallu ajouter des routes statiques ou un protocole de routage dynamique pour que les deux PCs puissent se connecter entre eux... Il faudrait également que les "routeurs" R1 et R2 soient définis comme étant passerelle par défaut pour leurs PCs respectifs.

D - IPv6 : prise en main

Voici donc l'affichage des adresses IP configurées sur le PC1 lors de la saisie de la commande *ip address* :

```
pc1:~# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: teql0: <NOARP> mtu 1500 qdisc noop qlen 100
    link/void
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
 qlen 1000
    link/ether 6e:5f:98:37:0c:07 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 scope global eth0
    inet6 fe80::6c5f:98ff:fe37:c07/64 scope link
       valid_lft forever preferred_lft forever
```

Et sinon nous voulons seulement les adresses de type IPv6 et sur l'interface lo, nous pouvons utiliser la commande *ip -6 addr ls dev lo* :

```
pc1:~# ip -6 addr ls dev lo
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

L'adresse IPv6 donnée à l'interface lo est donc ::1/128.

Pour envoyer trois paquets Internet Control Message Protocol Version 6 (ICMPv6) vers l'interface de loopback en IPv6, nous pouvons utiliser la commande *ping6 -c 3 ::1*:

```
pc1:~# ping6 -c 3 ::1
PING ::1(::1) 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.054 ms
64 bytes from ::1: icmp_seq=2 ttl=64 time=0.043 ms
64 bytes from ::1: icmp_seq=3 ttl=64 time=0.047 ms

--- ::1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 0.043/0.048/0.054/0.004 ms
```

Et trois paquets ICMPv6 ont effectivement été envoyés avec succès à l'adresse ::1/128 qui est donc l'interface de loopback du PC1. Il faut donc utiliser la commande *ping6* au lieu de *ping* pour les adresses IPv6.

Et pour connaître le(s) nom(s) associé(s) à cette adresse de loopback, nous pouvons afficher le contenu du fichier /etc/hosts au moyen de la commande *cat /etc/hosts* :

```
pc1:~# cat /etc/hosts
127.0.0.1 pc1
127.0.0.1 localhost localhost.localdomain

::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Et les noms associés à cette adresse IPv6 sont ip6-localhost et ip6-loopback.

Et sinon nous voulons seulement les adresses de type IPv6 et sur l'interface eth0, nous pouvons utiliser la commande *ip -6 addr ls dev eth0* :

```
pc1:~# ip -6 addr ls dev eth0
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 1000
    inet6 fe80::6c5f:98ff:fe37:c07/64 scope link
       valid_lft forever preferred_lft forever
```

Et nous observons donc l'adresse IPv6 configurée sur l'interface eth0 qui est donc fe80::6c5f:98ff:fe37:c07/64. Une adresse qui est donc une adresse unicast link-local d'après son préfixe fe80.

Information confirmée par l'utilisation de la commande *ipv6calc -qi fe80::6c5f:98ff:fe37:c07/64* qui affiche les informations relatives à l'adresse IPv6 saisie :

```
pc1:~# ipv6calc -qi fe80::6c5f:98ff:fe37:c07/64
Address type: unicast, link-local
Registry for address: reserved
Interface identifier: 6c5f:98ff:fe37:0c07
EUI-48/MAC address: 6e:5f:98:37:0c:07
MAC is a local one
MAC is an unicast one
```

Et la commande *ip address | grep -i scope* permet, comme la commande ip address d'afficher les différentes adresses IP sur toutes les interfaces mais en affichant que les lignes contenant le terme "scope" (ce qui permet de simplement afficher les adresses IP et leurs interfaces) :

```
pc1:~# ip address | grep -i scope
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.0.0.1/24 scope global eth0
    inet6 fe80::6c5f:98ff:fe37:c07/64 scope link
```

Et l'adresse link-local associée à l'adresse Media Access Control (MAC) 00:4f:4e:08:25:1a ressemblerait donc à fe80::24f:4eff:fe08:251a…

3 - PARAMÉTRAGE ET ANALYSE DES PROPRIÉTÉS IPV6

E - IPv6 : paramétrage IP Statique de base

Nous pouvons à présent configurer les adresses IPv6, nous allons donc ajouter une adresse IPv6 à l'interface eth0 du poste PC1 au moyen de la commande *ip addr add 2001:db8:46::1/64 dev eth0* puis vérifier l'ajout de cette adresse avec la commande *ip address* :

```
pc1:~# ip addr add 2001:db8:46::1/64 dev eth0
pc1:~# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: teql0: <NOARP> mtu 1500 qdisc noop qlen 100
    link/void
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
 qlen 1000
    link/ether 6e:5f:98:37:0c:07 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/24 scope global eth0
    inet6 2001:db8:46::1/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::6c5f:98ff:fe37:c07/64 scope link
       valid_lft forever preferred_lft forever
```

Et l'adresse IPv6 a bel et bien été ajoutée, nous remarquons que c'est une adresse unicast globale grâce à son préfixe étant 2001:db8.

Nous pouvons également afficher la table de routage du PC1 au moyen de la commande *route -n -A inet6* sachant que l'option *-n* permet d'afficher l'adresse en numérique au lieu du nom et l'option *-A* la famille d'adresse inet ou inet6 :

```
pc1:~# route -n -A inet6
Kernel IPv6 routing table
Destination                    Next Hop                   Flag Met R
ef Use If
2001:db8:46::/64               ::                         U    256 0
     0 eth0
fe80::/64                      ::                         U    256 0
     0 eth0
::/0                           ::                         !n   -1  1
     1 lo
::1/128                        ::                         Un   0   1
     7 lo
2001:db8:46::1/128             ::                         Un   0   1
     0 lo
fe80::6c5f:98ff:fe37:c07/128   ::                         Un   0   1
     0 lo
ff00::/8                       ::                         U    256 0
     0 eth0
::/0                           ::                         !n   -1  1
     1 lo
```

Cette table de routage configure PC1 pour communiquer au sein du lien local IPv6 et de ses propres adresses spécifiques, mais sans accès routé vers d'autres réseaux…

La commande *netstat -natup -A inet6* affiche une liste des connexions réseau actives, des ports ouverts, et des programmes qui les utilisent, spécifiquement pour le protocole IPv6. Voici le détail de chaque option :

- L'option *-n* affiche les adresses IP sous forme numérique, sans tenter de les résoudre en noms d'hôtes, ce qui rend l'affichage plus rapide.
- L'option *-a* montre toutes les connexions réseau, y compris celles en écoute (listening).
- L'option *-t* filtre pour afficher uniquement les connexions Transfert Control Protocol (TCP).
- L'option *-u* filtre pour afficher uniquement les connexions User Datagram Protocol (UDP).
- L'option *-p* associe chaque connexion ou socket affichée au nom et au Process Identifier (PID) du processus qui l'utilise (nécessite des privilèges administrateur).
- L'option *-A inet6* limite l'affichage aux adresses IPv6.

Cette commande ne détecte pour le moment rien de particulier :

```
pc1:~# netstat -natup -A inet6
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         St
ate       PID/Program name
```

Cependant, en lançant la commande */etc/init.d/ssh start*, nous lançons le service Secure Shell (SSH) sur le système. Elle initialise le démon SSH, permettant des connexions sécurisées à distance vers la machine. En démarrant SSH, le système devient accessible pour des sessions distantes via le protocole SSH, ce qui est essentiel pour l'administration et la gestion à distance. Et cela va donc générer des connexions actives qui seront cette fois ci affichées par la même commande que précédemment :

```
pc1:~# /etc/init.d/ssh start
Starting OpenBSD Secure Shell server: sshd.
pc1:~# netstat -natup -A inet6
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         St
ate       PID/Program name
tcp6       0      0 :::22                   :::*                    LI
STEN      525/sshd
```

Nous observons donc que le PC1 est en écoute sur le port 22, prêt à accepter les connexions SSH entrantes…

A noter que les adresses IPv6 unicast peuvent être regroupées en deux catégories : les adresses unicast link-local et les adresses unicast globales.

F - IPv6 : Adresse Multicast

Intéressons nous de plus près aux adresses IPv6 multicast du PC1, et pour cela, affichons le contenu du fichier /etc/hosts :

```
pc1:~# cat /etc/hosts
127.0.0.1 pc1
127.0.0.1 localhost localhost.localdomain

::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
```

Nous remarquons que les cinq dernières adresses IPv6 contenues dans le fichier /etc/hosts sont des adresses multicast réservées pour des groupes spécifiques de nœuds sur le réseau. Avec fe00::0 étant destinée au réseau local, tandis que ff00::0 définit un préfixe pour les adresses multicast. ff02::1 cible tous les nœuds sur le lien local, permettant à tous les appareils de recevoir certains paquets multicast. ff02::2 s'adresse à tous les “routeurs” du lien local, permettant la communication entre “routeurs”. Enfin, ff02::3 concerne tous les hôtes, facilitant l'envoi de messages à tous les appareils d’un réseau local.

Nous pouvons également afficher tous les abonnements multicast du PC1 au moyen de la commande netstat -g6n en montrant les adresses en format numérique en utilisant l'option -n pour éviter la résolution des noms, et en se limitant au groupe multicast du protocole IPv6 -g6 :

```
pc1:~# netstat -g6n
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         St
ate
IPv6/IPv4 Group Memberships
Interface       RefCnt Group
--------------- ------ ---------------------
lo              1      ff02::1
teql0           1      ff02::1
eth0            1      ff02::1:ff00:1
eth0            1      ff02::1:ff37:c07
eth0            1      ff02::1
```

Et nous avons donc les interfaces lo, teql0, et eth0 qui sont abonnées à ff02::1, l'adresse multicast pour tous les nœuds sur le lien local. L'interface eth0 est également abonné à ff02::1:ff00:1 et ff02::1:ff37:c07, des adresses de sollicitation de voisinage utilisées pour découvrir les appareils voisins par leur adresse MAC. Ces abonnements permettent de recevoir des messages multicast essentiels pour la communication IPv6 sur le réseau local.

Un recherche que nous aurions pu limiter à l'interface eth0 au moyen de la commande *ip -6 maddress show dev eth0* :

```
pc1:~# ip -6 maddress show dev eth0
3:      eth0
        inet6 ff02::1:ff00:1
        inet6 ff02::1:ff37:c07
        inet6 ff02::1
```

G - IPv6 : Le protocole

A présent, configurons une adresse IP sur l'interface eth0 du "routeur" R1 au moyen de la commande *ip addr add 2001:db8:46::2/64 dev eth0* :

```
r1:~# ip addr add 2001:db8:46::2/64 dev eth0
r1:~# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: teql0: <NOARP> mtu 1500 qdisc noop qlen 100
    link/void
3: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast q
len 1000
    link/ether 0e:ab:f8:0c:10:4b brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.2/24 scope global eth0
    inet6 2001:db8:46::2/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::cab:f8ff:fe0c:104b/64 scope link
       valid_lft forever preferred_lft forever
4: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast q
len 1000
    link/ether fa:de:dc:30:96:57 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.1/24 scope global eth1
    inet6 fe80::f8de:dcff:fe30:9657/64 scope link
       valid_lft forever preferred_lft forever
```

Et l'adresse a bel et bien été ajoutée… Aucune raison qu'un message ICMP entre R1 et le PC1 ne soit pas transmis correctement, nous utiliserons la commande *ping6 -c 1 2001:db8:46::1* :

```
r1:~# ping6 -c 1 2001:db8:46::1
PING 2001:db8:46::1(2001:db8:46::1) 56 data bytes
64 bytes from 2001:db8:46::1: icmp_seq=1 ttl=64 time=11.7 ms

--- 2001:db8:46::1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 11.764/11.764/11.764/0.000 ms
```

Et le paquet est transmis correctement…

Refaisons de même mais avec 4 paquets et en effectuant une capture de trames au moyen de la commande tcpdump -i eth0 ip6 -w /hosthome/r1pc1.pcap qui permettra donc de capturer les trames IPv6 (pour l'option ip6) de l'interface eth0 (pour l'option -i eth0) et de l'écrire dans un fichier /hosthome/r1pc1.pcap (pour l'option -w suivie du nom du fichier) :

```
r1:~# ping6 -c 4 2001:db8:46::1
PING 2001:db8:46::1(2001:db8:46::1) 56 data bytes
64 bytes from 2001:db8:46::1: icmp_seq=1 ttl=64 time=3.62 ms
64 bytes from 2001:db8:46::1: icmp_seq=2 ttl=64 time=0.402 ms
64 bytes from 2001:db8:46::1: icmp_seq=3 ttl=64 time=0.639 ms
64 bytes from 2001:db8:46::1: icmp_seq=4 ttl=64 time=0.952 ms

--- 2001:db8:46::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3009ms
rtt min/avg/max/mdev = 0.402/1.403/3.620/1.294 ms
```

```
pc1:~# tcpdump -i eth0 ip6 -w /hosthome/r1pc1.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size
96 bytes
^C16 packets captured
16 packets received by filter
0 packets dropped by kernel
```

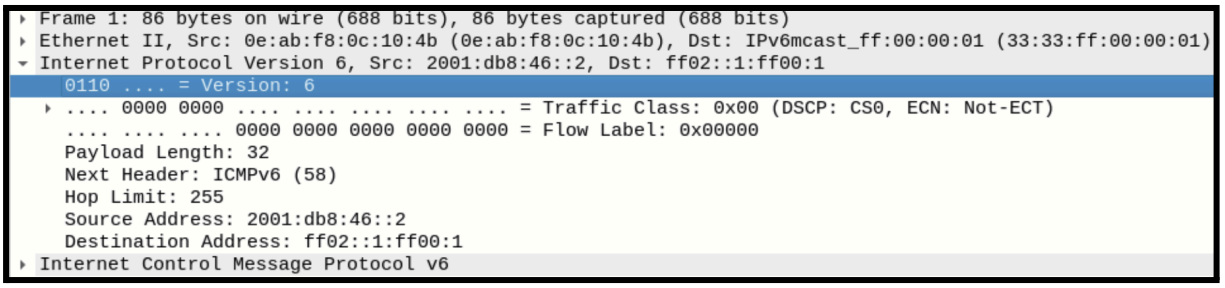
Ouvrons le fichier créé avec Wireshark :

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	2001:db8:46::2	ff02::1:ff00:1	ICMPv6	86	Neighbor Solicitation for 2001:db8:46::1 from 0e:ab:f8:0c:10:4b
2	0.000022	2001:db8:46::1	2001:db8:46::2	ICMPv6	86	Neighbor Advertisement 2001:db8:46::1 (sol, ovr) is at 6e:5f:98:37:0c:07
3	0.000057	2001:db8:46::2	2001:db8:46::1	ICMPv6	118	Echo (ping) request id=0x0a02, seq=1, hop limit=64 (reply in 4)
4	0.000062	2001:db8:46::1	2001:db8:46::2	ICMPv6	118	Echo (ping) reply id=0x0a02, seq=1, hop limit=64 (request in 3)
5	1.005620	2001:db8:46::2	2001:db8:46::1	ICMPv6	118	Echo (ping) request id=0x0a02, seq=2, hop limit=64 (reply in 6)
6	1.005657	2001:db8:46::1	2001:db8:46::2	ICMPv6	118	Echo (ping) reply id=0x0a02, seq=2, hop limit=64 (request in 5)
7	2.005513	2001:db8:46::2	2001:db8:46::1	ICMPv6	118	Echo (ping) request id=0x0a02, seq=3, hop limit=64 (reply in 8)
8	2.005557	2001:db8:46::1	2001:db8:46::2	ICMPv6	118	Echo (ping) reply id=0x0a02, seq=3, hop limit=64 (request in 7)
9	3.006372	2001:db8:46::2	2001:db8:46::1	ICMPv6	118	Echo (ping) request id=0x0a02, seq=4, hop limit=64 (reply in 10)
10	3.006408	2001:db8:46::1	2001:db8:46::2	ICMPv6	118	Echo (ping) reply id=0x0a02, seq=4, hop limit=64 (request in 9)
11	4.995591	fe80::6c5f:98ff:fe3…	2001:db8:46::2	ICMPv6	86	Neighbor Solicitation for 2001:db8:46::2 from 6e:5f:98:37:0c:07
12	4.995796	2001:db8:46::2	fe80::6c5f:98ff:fe3…	ICMPv6	78	Neighbor Advertisement 2001:db8:46::2 (sol)
13	10.005407	fe80::cab:f8ff:fe0c…	fe80::6c5f:98ff:fe3…	ICMPv6	86	Neighbor Solicitation for fe80::6c5f:98ff:fe37:c07 from 0e:ab:f8:0c:10:4b
14	10.005436	fe80::6c5f:98ff:fe3…	fe80::cab:f8ff:fe0c…	ICMPv6	78	Neighbor Advertisement fe80::6c5f:98ff:fe37:c07 (sol)
15	14.995187	fe80::6c5f:98ff:fe3…	fe80::cab:f8ff:fe0c…	ICMPv6	86	Neighbor Solicitation for fe80::cab:f8ff:fe0c:104b from 6e:5f:98:37:0c:07
16	14.995395	fe80::cab:f8ff:fe0c…	fe80::6c5f:98ff:fe3…	ICMPv6	78	Neighbor Advertisement fe80::cab:f8ff:fe0c:104b (sol)

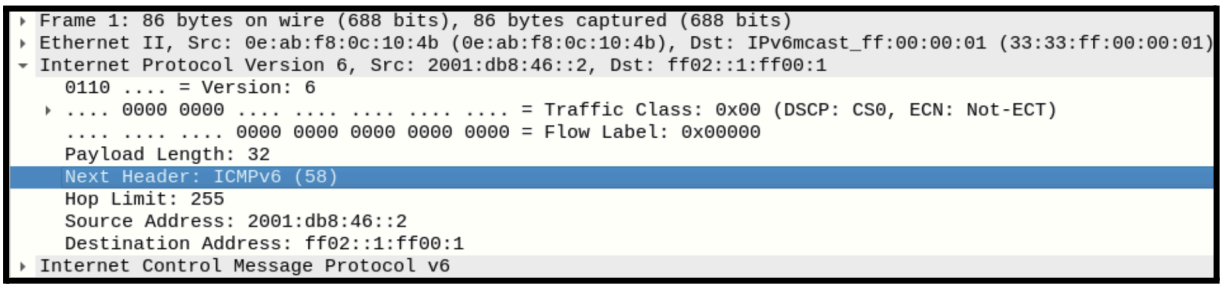
Nous observons qu'il contient 16 trames ICMPv6, chose confirmée par le champ Type de ces trames qui est IPv6 sachant qu'il identifie le protocole de la couche réseau auquel les données de la trame doivent être transmises :

```
▸ Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits)
▾ Ethernet II, Src: 0e:ab:f8:0c:10:4b (0e:ab:f8:0c:10:4b), Dst: IPv6mcast_ff:00:00:01 (33:33:ff:00:00:01)
    ▸ Destination: IPv6mcast_ff:00:00:01 (33:33:ff:00:00:01)
    ▸ Source: 0e:ab:f8:0c:10:4b (0e:ab:f8:0c:10:4b)
      Type: IPv6 (0x86dd)
▸ Internet Protocol Version 6, Src: 2001:db8:46::2, Dst: ff02::1:ff00:1
▸ Internet Control Message Protocol v6
```

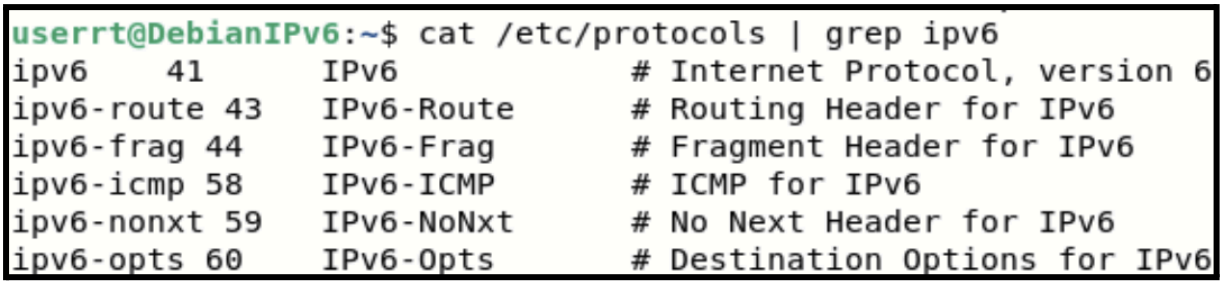
Une chose également confirmée par le champ Version des paquets IP échangés qui prend la valeur Version 6 donc pour IPv6 :



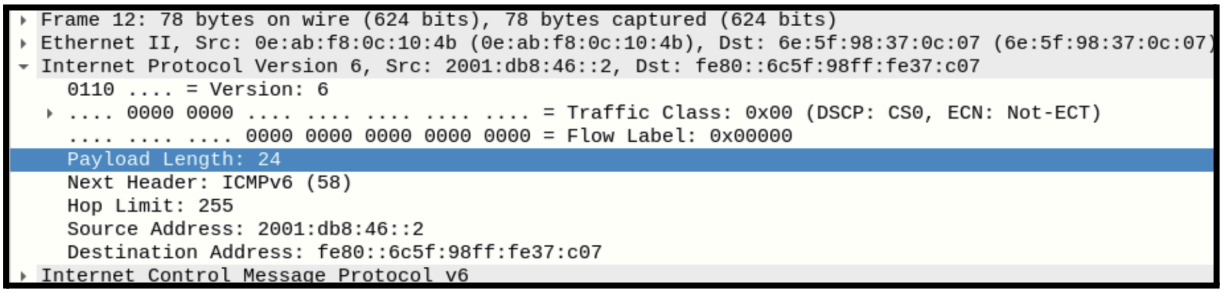
Pour tous les paquets, le champ Next Header indique le type du prochain en-tête ou protocole qui suit l'en-tête IPv6 et prend logiquement la valeur 58 indiquant une en-tête ICMPv6 :

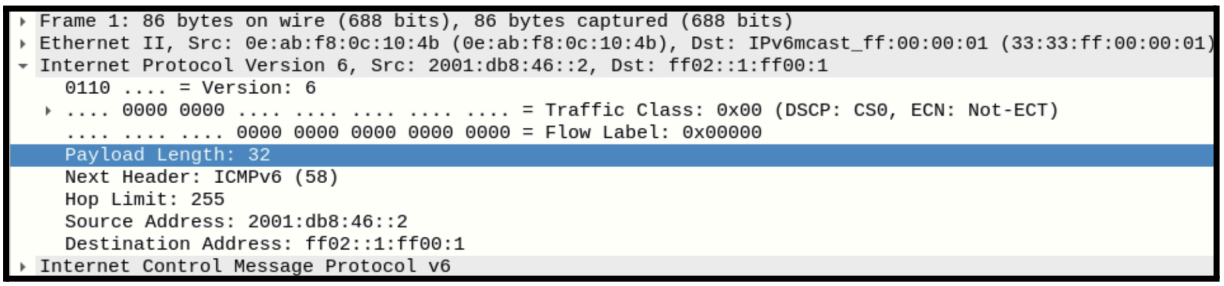


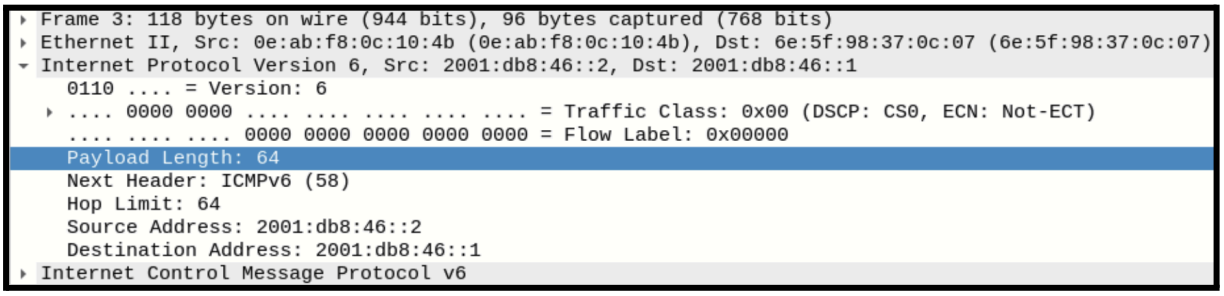
Une association valeur/protocole confirmé par le contenu du fichier /etc/protocols, nous affichons le contenu au moyen de la commande *cat /etc/protocols | grep ipv6* sachant que *grep ipv6* permettra de sélectionner uniquement les lignes contenant le terme IPv6 afin de simplifier la recherche :



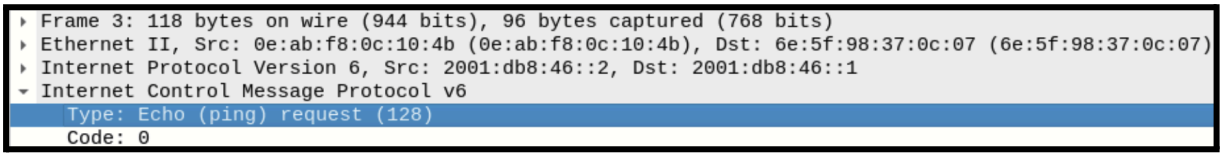
Le champ Payload Length dans les paquets IPv6 indique la taille, en octets, des données contenues après l'en-tête principal IPv6, et, dans notre cas, prend des valeurs qui varient en fonction des paquets, entre 24, 32 et 64 octets :

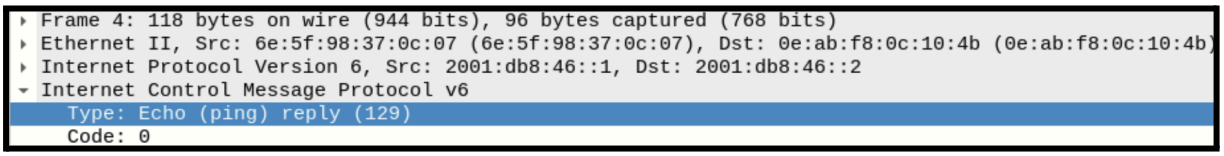


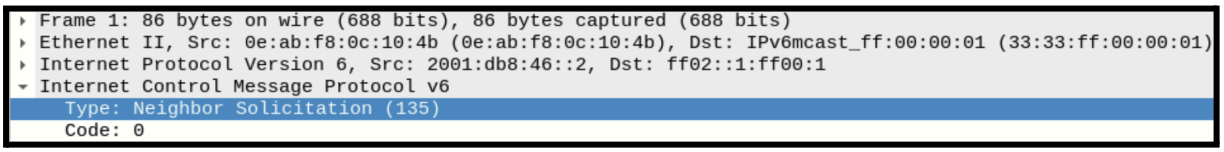


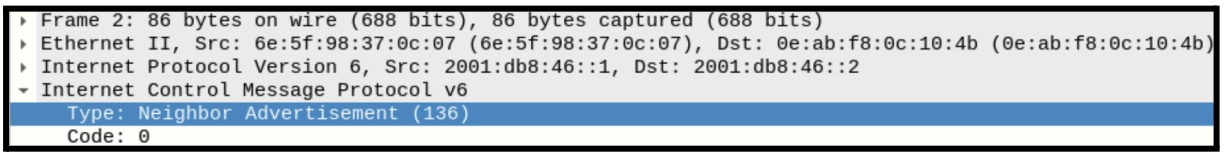


Le type d'un message ICMP permet de le catégoriser en fonction de sa nature, tandis que le code fournit encore plus d'informations supplémentaires sur sa nature. Dans nos 16 trames, nous distinguons 4 valeurs possible de type et une seule de code : les valeurs 128, 129, 135 et 136, signifiant dans l'ordre Echo Ping Request, Echo Ping Reply, Neighbor Solicitation et Neighbor Advertisement, toutes associées au code 0 qui dans ces cas-cis, signifie qu'il n'y a pas d'informations supplémentaires à ajouter :









Nous remarquons que contrairement à l'Internet Control Message Protocol Version 4 (ICMPv4), il n'y a ni trames du protocole Address Resolution Protocol (ARP) remplacées par le Neighbor Discovery Protocol (NDP), ni adresses broadcast qui n'existent pas en IPv6, remplacées par les adresses multicast. Chose mise en avant lors de la première trame et de son adresse de destination étant une adresse IPv6 multicast utilisée dans le cadre du NDP :

Les paquets ICMPv6 Neighbor Solicitation ont pour rôle principal de permettre la découverte des voisins dans un réseau IPv6, afin de résoudre les adresses IPv6 en adresses MAC sur le lien local.

Puis nous pouvons consulter le cache des voisins sur les machines PC1 et R1 avec la commande *ip neigh show* :

```
pc1:~# ip neigh show
2001:db8:46::2 dev eth0 lladdr 0e:ab:f8:0c:10:4b DELAY
```

```
r1:~# ip neigh show
fe80::6c5f:98ff:fe37:c07 dev eth0 lladdr 6e:5f:98:37:0c:07 STALE
2001:db8:46::1 dev eth0 lladdr 6e:5f:98:37:0c:07 REACHABLE
```

L'adresse IPv6 globale de R1 est donc reconnue par PC1 et idem pour PC1 envers R1 en plus de l'adresse IP unicast lien-local... Les différents états tels que DELAY, STALE ou encore REACHABLE que nous pouvons apercevoir sont biaisés car il faut envoyer une requête ping pendant l'affichage du cache pour que celui-ci ne soit pas vide, ils devraient tous être en REACHABLE.

4 - ROUTAGE STATIQUE IPV6

H - Routage statique IPv6

Sachant que toutes les interfaces Ethernet sont activées, nous pouvons les configurer pour tous les équipements selon ce tableau d'adressage :

Equipement et interface	**Adresse IPv6 globale configurée**
PC1 eth0	2001:db8:46::1/64
R1 eth0	2001:db8:46::2/64
R1 eth1	2001:db8:64::1/64
R2 eth0	2001:db8:64::2/64
R2 eth1	2001:db8:64:1::2/64
PC2 eth0	2001:db8:64:1::1/64

Configurons donc les adresses IP n'ayant pas encore été ajoutées :

```
r1:~# ip addr add 2001:db8:64::1/64 dev eth1
```

```
r2:~# ip addr add 2001:db8:64::2/64 dev eth0
```

```
r2:~# ip addr add 2001:db8:64:1::2/64 dev eth1
```

```
pc2:~# ip addr add 2001:db8:64:1::1/64 dev eth0
```

Mais les routes ne sont toujours pas configurées entre PC1 et PC2 :

```
pc1:~# route -n -A inet6
Kernel IPv6 routing table
Destination                    Next Hop                   Flag Met Ref
 Use If
2001:db8:46::/64               ::                         U    256 0
   0 eth0
fe80::/64                      ::                         U    256 0
   0 eth0
::/0                           ::                         !n   -1  1
  78 lo
::1/128                        ::                         Un   0   1
   1 lo
2001:db8:46::1/128             ::                         Un   0   1
  77 lo
fe80::6c5f:98ff:fe37:c07/128   ::                         Un   0   1
  11 lo
ff00::/8                       ::                         U    256 0
   0 eth0
::/0                           ::                         !n   -1  1
  78 lo
```

Activons donc le forwarding sur les deux “routeurs” R1 et R2 en utilisant les commandes *echo "1" > /proc/sys/net/ipv6/conf/default/forwarding* puis *echo "1" > /proc/sys/net/ipv6/conf/all/forwarding* :

```
r1:~# echo "1" > /proc/sys/net/ipv6/conf/default/forwarding
r1:~# echo "1" > /proc/sys/net/ipv6/conf/all/forwarding
```

```
r2:~# echo "1" > /proc/sys/net/ipv6/conf/default/forwarding
r2:~# echo "1" > /proc/sys/net/ipv6/conf/all/forwarding
```

Nous pouvons alors configurer des routes opérationnelles entre PC1 et PC2 au moyen des commandes *route -A inet6 add 2001:db8:64::/64 gw 2001:db8:46::2 dev eth0* et *route -A inet6 add 2001:db8:64:1::/64 gw 2001:db8:46::2 dev eth0* effectuées sur PC1, puis *route -A inet6 add 2001:db8:64:1::/64 gw 2001:db8:64::2 dev eth1* effectuée sur R1, puis *route -A inet6 add 2001:db8:46::/64 gw 2001:db8:64::1 dev eth0* effectuée sur R2 et enfin *route -A inet6 add 2001:db8:64::/64 gw 2001:db8:64:1::2 dev eth0* et *route -A inet6 add 2001:db8:46::/64 gw 2001:db8:64:1::2 dev eth0* effectuées sur PC2 :

```
pc1:~# route -A inet6 add 2001:db8:64::/64 gw 2001:db8:46::2 dev eth0
pc1:~# route -A inet6 add 2001:db8:64:1::/64 gw 2001:db8:46::2 dev eth
0

r1:~# route -A inet6 add 2001:db8:64:1::/64 gw 2001:db8:64::2 dev eth1

r2:~# route -A inet6 add 2001:db8:46::/64 gw 2001:db8:64::1 dev eth0

pc2:~# route -A inet6 add 2001:db8:64::/64 gw 2001:db8:64:1::2 dev eth
0
pc2:~# route -A inet6 add 2001:db8:46::/64 gw 2001:db8:64:1::2 dev eth
0
```

Et voici donc les tables de routage de tous les équipements uniquement sur les interfaces concernées par nos configurations obtenus au moyen de la commande *ip -6 route list dev ethX*, le X étant à adapter et en commençant par PC1 :

```
pc1:~# ip -6 route list dev eth0
2001:db8:46::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
2001:db8:64::/64 via 2001:db8:46::2  metric 1  mtu 1500 advmss 1440 ho
plimit 4294967295
2001:db8:64:1::/64 via 2001:db8:46::2  metric 1  mtu 1500 advmss 1440
hoplimit 4294967295
fe80::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
```

Puis R1 :

```
r1:~# ip -6 route list dev eth0
2001:db8:46::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
fe80::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
r1:~# ip -6 route list dev eth1
2001:db8:64::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
2001:db8:64:1::/64 via 2001:db8:64::2  metric 1  mtu 1500 advmss 1440
hoplimit 4294967295
fe80::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
```

Puis R2 :

```
r2:~# ip -6 route list dev eth0
2001:db8:46::/64 via 2001:db8:64::1  metric 1  mtu 1500 advmss 1440 ho
plimit 4294967295
2001:db8:64::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
fe80::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
r2:~# ip -6 route list dev eth1
2001:db8:64:1::/64  metric 256  mtu 1500 advmss 1440 hoplimit 42949672
95
fe80::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
```

Et enfin PC2 :

```
pc2:~# ip -6 route list dev eth0
2001:db8:46::/64 via 2001:db8:64:1::2  metric 1  mtu 1500 advmss 1440
hoplimit 4294967295
2001:db8:64::/64 via 2001:db8:64:1::2  metric 1  mtu 1500 advmss 1440
hoplimit 4294967295
2001:db8:64:1::/64  metric 256  mtu 1500 advmss 1440 hoplimit 42949672
95
fe80::/64  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
```

Pour tous les réseaux inconnus des différents équipements, il y a un route permettant d'y accéder qui a été ajoutée, il n'y a donc aucune raison pour que la connectivité de la topologie ne soit pas totale, testons la avec ces différents tests en commençant de PC1 vers PC2 :

```
pc1:~# ping6 -c 5 2001:db8:64:1::1
PING 2001:db8:64:1::1(2001:db8:64:1::1) 56 data bytes
64 bytes from 2001:db8:64:1::1: icmp_seq=1 ttl=62 time=53.4 ms
64 bytes from 2001:db8:64:1::1: icmp_seq=2 ttl=62 time=1.06 ms
64 bytes from 2001:db8:64:1::1: icmp_seq=3 ttl=62 time=0.578 ms
64 bytes from 2001:db8:64:1::1: icmp_seq=4 ttl=62 time=0.482 ms
64 bytes from 2001:db8:64:1::1: icmp_seq=5 ttl=62 time=1.35 ms

--- 2001:db8:64:1::1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4034ms
rtt min/avg/max/mdev = 0.482/11.390/53.469/21.041 ms
```

Puis de PC2 vers PC1 :

```
pc2:~# ping6 -c 5 2001:db8:46::1
PING 2001:db8:46::1(2001:db8:46::1) 56 data bytes
64 bytes from 2001:db8:46::1: icmp_seq=1 ttl=62 time=22.0 ms
64 bytes from 2001:db8:46::1: icmp_seq=2 ttl=62 time=0.235 ms
64 bytes from 2001:db8:46::1: icmp_seq=3 ttl=62 time=0.586 ms
64 bytes from 2001:db8:46::1: icmp_seq=4 ttl=62 time=0.537 ms
64 bytes from 2001:db8:46::1: icmp_seq=5 ttl=62 time=0.821 ms

--- 2001:db8:46::1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 0.235/4.842/22.032/8.597 ms
```

Et comme attendu, tout fonctionne parfaitement…

Nous avons donc pu mieux comprendre et mettre en pratique la configuration d'un réseau IPv6 avec plusieurs machines interconnectées. En configurant les interfaces réseau, en attribuant des adresses IPv6 appropriées et en ajoutant les routes nécessaires, nous avons pu établir une communication fluide entre les équipements, allant des PC aux routeurs. Ce processus a aussi illustré l'importance des commandes réseau telles que *ip*, *route* et *ping6* pour tester et maintenir la connectivité. Nos compétences en gestion de réseaux IPv6 et dans l'utilisation d'outils Linux pour le diagnostic et la configuration des infrastructures réseau ont donc été renforcées.

Pour une plus grande topologie, configurer les adresses IP ainsi que les routes de manière statique serait bien évidemment une perte de temps, il aurait donc pu être intéressant d'explorer l'implémentation et la gestion de réseaux IPv6 dans un environnement de production plus complexe, avec des protocoles de routage dynamique tels que Open Shortest Path First Version 3 (OSPFv3) ou Border Gateway Protocol (BGP) pour IPv6, afin d'assurer l'efficacité des communications. De plus, nous pourrions nous intéresser à la sécurité des réseaux IPv6 en étudiant les mécanismes intégrés comme Internet Protocol Security (IPSec), ainsi que la gestion des adresses et des sous-réseaux via des outils de gestion automatique tels que Dynamic Host Configuration Protocol Version 6 (DHCPv6) et Stateless Automatic Auto Configuration (SLAAC). Ces éléments nous auraient permis d'encore mieux comprendre les défis auxquels les administrateurs réseau sont confrontés dans la gestion de l'IPv6 à grande échelle…