Université d'Aix-Marseille

UFR des Sciences
Département d'Informatique

Projet Réseaux

Spécialité :

Master 1 Informatique

Science et Ingénierie des Données

Thème :

Communications entre îlots IPv4 au sein d'un monde IPv6

Réalisé par :

M^lle^ ZEMMOURI Yasmine.

M^lle^ BEKHADDA Hadjira.

Responsable :

M. GHANNOU Omar.

Année Universitaire : 2023-2024.

1. Introduction :

Le déploiement d'IPv6, la dernière version du protocole Internet, est devenu impératif en raison de l'épuisement imminent des adresses IPv4 disponibles. Cependant, la transition complète d'IPv4 à IPv6 peut poser des défis, car de nombreux réseaux et systèmes existants fonctionnent toujours sur IPv4. Pour surmonter cette transition tout en permettant une connectivité continue entre les réseaux IPv4 et IPv6, différentes techniques de tunneling ont été développées. Le tunneling IPv6 over IPv4, souvent désigné sous le nom de 6in4, est l'une de ces méthodes qui offre une solution efficace pour établir des connexions entre des réseaux IPv6 et IPv4 distincts.

Nous aborderons lors de ce projet les principes fondamentaux du tunneling, examinerons les détails de la création d'une interface tunnel, et discuterons des défis et des considérations importantes associés à cette approche. En outre, nous décrirons comment cette solution a été intégrée dans un scénario concret impliquant la création d'un tunnel entre un réseau IPv4 existant et un réseau IPv6 émergent.

2. Configuration réseau :

La configuration réseau constitue une étape cruciale dans la mise en œuvre réussie de notre projet visant à établir une connectivité bidirectionnelle entre des machines IPv4 via un tunnel. Nous avons mis en place un réseau, en déployant les configurations avec ansible. Ce dernier servira à déployer notre système.

Vérifions l'accessibilité entre machines, via des ping :

VM1 et VM1-6 :

VM1 vers VM1-6

```
m1reseaux@VM1:~$ ping 172.16.2.156
m1reseaux@VM1:~$ ping 172.16.2.156
PING 172.16.2.156 (172.16.2.156) 56(84) bytes of data.
64 bytes from 172.16.2.156: icmp_seq=1 ttl=64 time=0.670 ms
64 bytes from 172.16.2.156: icmp_seq=2 ttl=64 time=0.732 ms
64 bytes from 172.16.2.156: icmp_seq=3 ttl=64 time=0.825 ms
^C
--- 172.16.2.156 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.670/0.742/0.825/0.063 ms
```

VM1-6 et VM2-6 :

VM2-6 et VM3-6 :

VM1-6 et VM3-6 :

VM3-6 et VM3 :

```
                        VM3 VERS VM3-6

m1reseaux@VM3:/vagrant$ ping 172.16.2.186
PING 172.16.2.186 (172.16.2.186) 56(84) bytes of data.
64 bytes from 172.16.2.186: icmp_seq=1 ttl=64 time=1.07 ms
64 bytes from 172.16.2.186: icmp_seq=2 ttl=64 time=0.776 ms
64 bytes from 172.16.2.186: icmp_seq=3 ttl=64 time=0.740 ms
64 bytes from 172.16.2.186: icmp_seq=4 ttl=64 time=0.636 ms
64 bytes from 172.16.2.186: icmp_seq=5 ttl=64 time=0.711 ms
^C
--- 172.16.2.186 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10137ms
rtt min/avg/max/mdev = 0.636/0.734/1.071/0.111 ms
```

VM1 et VM3 :

- ➢ Une observation attentive de la connectivité entre les différentes machines révèle un modèle cohérent : toutes les machines sont accessibles les unes aux autres, sauf dans un cas particulier, à savoir la communication entre VM1 et VM3.
- ✓ Le résultat obtenu est conforme à nos attentes, car le ping direct entre une machine possédant une adresse IPv4 vers une autre n'est pas possible en l'absence d'une route établie entre ces deux machines. Cela met en évidence le besoin crucial de mettre en place une solution capable de créer un "tunnel" pour acheminer le trafic IPv4 via un chemin de type IPv6.
- ✓ L'objectif principal du projet est de remédier à l'absence de connectivité directe entre VM1 et VM3 en établissant un tunnel IPv6. Ce tunnel agira comme une infrastructure permettant le passage du trafic IPv4 entre ces deux machines, en le faisant transiter par le réseau IPv6 existant (VM1-6, VM2-6, et VM3-6).

3. L’interface virtuelle TUN :

Comme mentionné précédemment, l’objectif principal est d'établir une connexion fluide entre deux îlots IPv4 distincts en utilisant le réseau IPv6 comme une infrastructure de liaison. Cette initiative vise à surmonter les défis de la communication entre des réseaux isolés, offrant ainsi une solution robuste pour le passage de données entre ces deux entités distinctes.

La mise en œuvre d'un tunnel IPv4 sur IPv6 se positionne comme le mécanisme clé pour faciliter le transit des paquets IPv4 à travers le réseau IPv6. Cette méthode ingénieuse implique l'encapsulation des paquets IPv4 dans des paquets IPv6 lors de leur transit à travers le réseau intermédiaire. À la sortie du tunnel, les paquets sont désencapsulés, retrouvant ainsi leur format d'origine.

3.1. Création et configuration de l'interface :

Les étapes nécessaires à la création et à la configuration de l'interface tun0 sont fondamentales pour garantir une connectivité fluide et efficace entre ces deux environnements réseau distincts.

Cet ensemble d'étapes, allant de l'allocation initiale de l'interface à sa configuration détaillée et à son activation, est essentiel pour établir un lien opérationnel entre les îlots IPv4.

Afin de mener la création de l'interface à bien, l'allocation est la première étape. . Elle est effectuée en appelant la fonction tun_alloc, qui utilise le fichier spécial /dev/net/tun. Cela garantit que le noyau alloue une interface tunnel disponible. Une fois l'interface obtenue, elle est associée au descripteur de fichier tunfd, prêt à être configuré.

Cependant, malgré des efforts soutenus, une problématique persistait : l'allocation de l'interface tun0 ne se déroulait pas comme prévu, et le descripteur de fichier tunfd était systématiquement égal à -1, même avec un code apparemment correct. Après une analyse approfondie, notamment par la consultation de la documentation et l'exploration de différentes alternatives, la source du problème a été identifiée.

Il s'est avéré que le module responsable du chargement des interfaces virtuelles sur le noyau, nommé "modprobe", n'était pas activé. Ce module est essentiel pour garantir le bon fonctionnement de la création de l'interface tun0. La résolution de cette problématique a été trouvée en exécutant la commande système sudo modprobe tun. Cette commande a activé le module "modprobe", permettant ainsi une allocation correcte de l'interface tun0.

Après l'allocation, l'interface tun0 doit être configurée avec une adresse IP et d'autres paramètres réseau.

Un fichier bash « configure-tun.sh » est utilisée à cet effet. La commande ip addr add 172.16.2.1/28 dev tun0 configure l'interface avec l'adresse IP spécifiée et un masque de sous-réseau.

L'activation de l'interface tun0 est impérative pour qu'elle soit opérationnelle. La commande ip link set est utilisée pour activer l'interface nouvellement configurée. Par conséquent, l'exécution de configure-sh permet également à l'interface d'être prête à transmettre le trafic réseau.

Les routes sont essentielles pour diriger le trafic à travers l'interface tun0. Utilisant la commande ip route add, des règles de routage sont établies. Par exemple, ip route add 172.16.2.144 dev tun0 ajoute une route spécifique pour acheminer le trafic vers une destination particulière via l'interface tun0.

De plus, après la disparition de VM2, les informations de routage sont modifiées sur VM1 et VM1-6 : il suffit d'enlever les routes vers les LANS non accessibles, et rajouter les routes spécifiques à tun0, comme mentionné précedemment.

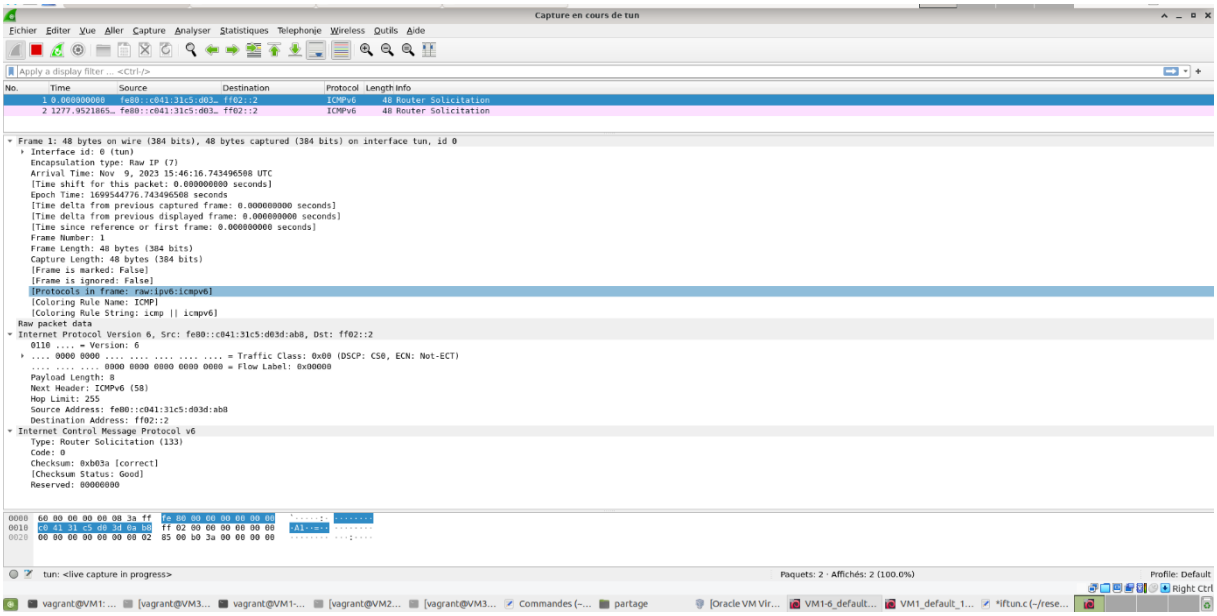
Enfin, une vérification de la configuration est effectuée pour s'assurer que toutes les étapes précédentes ont été réalisées avec succès. La commande ifconfig ou ip addr show est utilisée pour afficher les détails de l'interface tun0, y compris son adresse IP, son état opérationnel, et d'autres paramètres. Par exemple, system("ip addr show tun0"); permet de confirmer que l'interface a été correctement créée et configurée.

- Test du ping 172.16.2.1 :

```
m1reseaux@VM1-6:~$ ping 172.16.2.1
PING 172.16.2.1 (172.16.2.1) 56(84) bytes of data.
64 bytes from 172.16.2.1: icmp_seq=1 ttl=64 time=0.089 ms
64 bytes from 172.16.2.1: icmp_seq=2 ttl=64 time=0.054 ms
64 bytes from 172.16.2.1: icmp_seq=3 ttl=64 time=0.048 ms
64 bytes from 172.16.2.1: icmp_seq=4 ttl=64 time=0.047 ms
64 bytes from 172.16.2.1: icmp_seq=5 ttl=64 time=0.049 ms
^C
--- 172.16.2.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4078ms
rtt min/avg/max/mdev = 0.047/0.057/0.089/0.016 ms
```

Tous les paquets de la séquence du ping sont transmis avec succès. Cette réussite indique que les échanges de données s'effectuent sans perte notable.

Cependant, en capturant le trafic sur WireShark, comme montré sur la figure suivante :



Les premières trames capturées sont des requêtes ICMP Router Solicitation (type 133). Ce type de requête est envoyée pour demander des informations sur les routes disponibles.

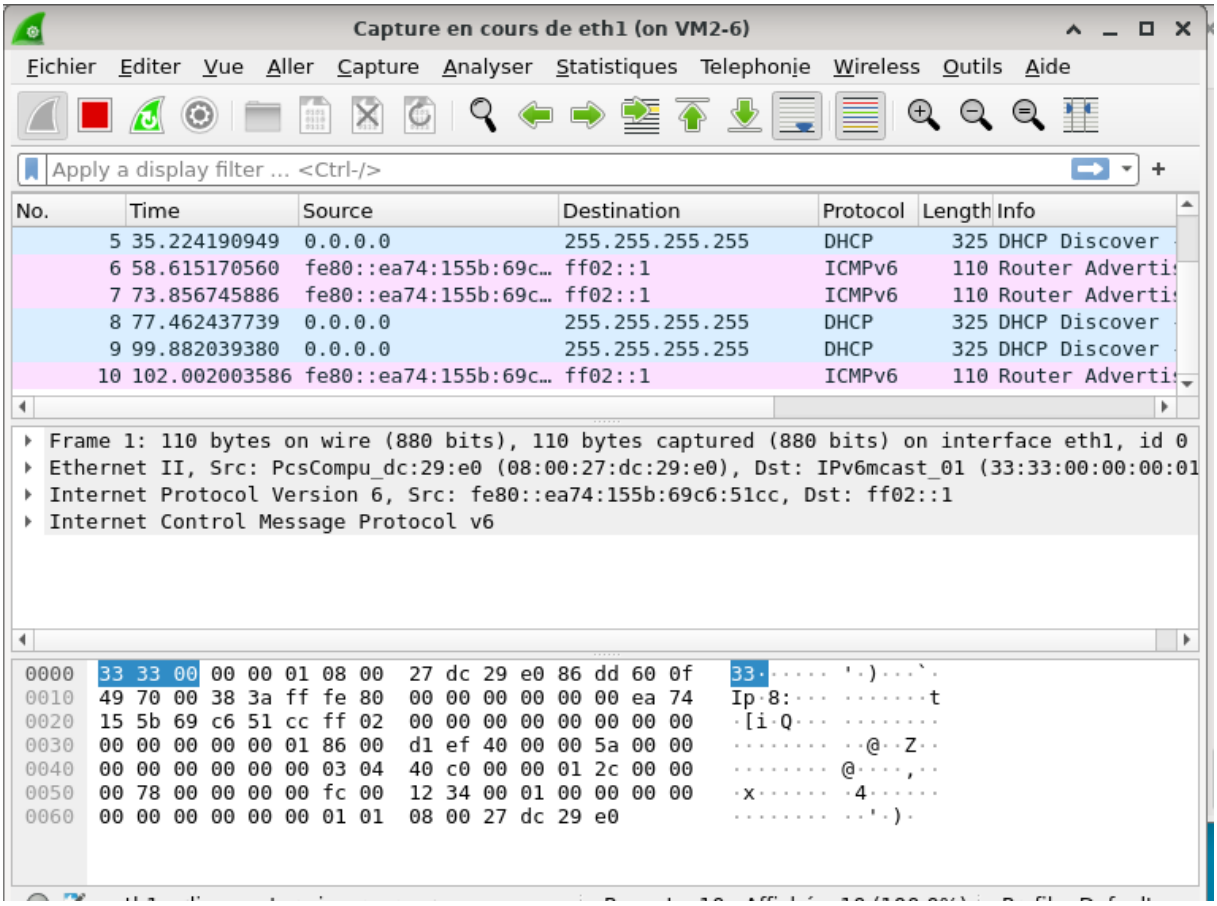
WireShark observe le trafic qui traverse une interface réseau. Le trafic s'arrete à tun0 :

Lorsqu'un ping est effectué sur l'interface elle-meme, que la capture Wireshark révèle uniquement des trames ICMP Router Solicitation (type 133) sans réponses ICMP Router Advertisement associées, cela peut être attribué à des mécanismes d'optimisation du processus de ping local. La machine elle-même peut traiter les requêtes ICMP Router Solicitation localement sans émettre de réponses ICMP Router Advertisement, ce qui n'est pas capturé dans la fenêtre observée par Wireshark.

- Test du ping 172.16.2.10 :

```
m1reseaux@VM1-6:~$ ping 172.16.2.10
PING 172.16.2.10 (172.16.2.10) 56(84) bytes of data.
```

Le ping est en attente, jusqu'à expiration du délai. On peut observer le trafic sur wireshark :



Dans ce cas précis, les paquets traversent l'interface tun0 à la recherche d'un destinataire, mais n'en trouve aucun :

Quand le ping est bloqué en attente d'une réponse à une requête ICMP Router Solicitation vers une adresse qui n'a pas de machine correspondante, cela est attribué à la nature même du protocole ICMP.

Lorsqu'une machine émet une requête ICMP Router Solicitation pour découvrir les routeurs disponibles sur le réseau, elle s'attend à recevoir des réponses ICMP Router Advertisement des routeurs actifs. Cependant, dans un contexte où aucune machine ne fonctionne comme routeur IPv6 actif, aucune réponse ne sera générée. En conséquence, le ping vers une adresse inexistante peut sembler bloqué en attente de réponses qui ne seront jamais reçues.

Aucune réponse ICMP Router Advertisement ne sera générée, et le ping restera en attente jusqu'à ce qu'il atteigne le délai d'expiration.

3.2. Récupération des paquets :

Lors de la création de l'interface tun0, un descripteur de fichier est renvoyé par la fonction tun_alloc, comme illustré précédemment. Ce descripteur de fichier est essentiel pour établir la communication avec l'interface TUN. Il permet notamment de lire les données qui sont envoyées à l'interface par le biais d'un appel à la fonction read.

Afin de compléter la bibliothèque iftun, une nouvelle fonction peut être introduite, prenant deux descripteurs de fichiers en paramètres, src et dst. La fonction, conçue pour opérer dans le contexte d'une interface TUN identifiée par le descripteur src, a pour objectif de copier de manière perpétuelle toutes les données lisibles sur src vers le fichier décrit par dst. En d'autres termes, elle établit un flux, permettant la transmission continue des données entre l'interface TUN et le fichier associé.

```
int copier(int src, int dst){
  if (src==-1) {
    perror("src vide");
    return -1;
  }

   if (dst==-1) {
    perror("dst vide");
    return -1;
  }

  char buff[1500];

  ssize_t n = 0;

  n = read(src, &buff, 1500);
  if (n<0) {
     perror("read retour negatif");
     return n;
  }

  buff[n] = '\0';
  write(dst, &buff, n);

  return n;
}
```

La fonction `copy_tun` est conçue pour effectuer une copie bidirectionnelle de données entre deux descripteurs de fichiers, identifiés par les paramètres `src` (source) et `dst` (destination).

Elle utilise une boucle infinie pour lire continuellement les données du descripteur source et les écrire dans le descripteur destination. De plus, elle assure une copie complète en vérifiant que la quantité d'octets lus est égale à celle écrite.

En testant le programme avec un ping 172.16.2.1 (ou 172.16.2.10), le programme semble fonctionner comme prévu, avec l'affichage des paquets échangés dans le terminal correspondant aux mêmes paquets observés dans une capture Wireshark. Cela confirme que les données transmises à travers l'interface TUN sont correctement capturées.

Cette correspondance entre la sortie standard et la capture Wireshark indique que les paquets sont transmis avec succès à travers le tunnel, et le programme semble accomplir la tâche de copie entre les descripteurs de fichiers associés à l'interface TUN et à la sortie standard.

- L'option `IFF_NO_PI` lors de la création d'une interface TUN indique au noyau Linux de ne pas inclure les informations d'en-tête du paquet lors de la lecture ou de l'écriture de données sur l'interface. Ces informations d'en-tête comprennent généralement des métadonnées telles que le protocole du paquet et sa longueur. L'ajout de cette option est utile lorsque l'application manipule directement les données du paquet sans nécessiter ces métadonnées. Ce qui peut simplifier le traitement des données.

4. Un tunnel simple pour IPv4 :

Dans la mise en place d'un tunnel IPv4 sur IPv6, un scénario classique consiste à établir une communication transparente entre deux réseaux IPv4 distincts en utilisant le réseau IPv6 comme canal de transport. Ce processus implique la création d'interfaces tunnel de part et d'autre du réseau IPv6, agissant comme des points d'entrée et de sortie du tunnel.

Pour réaliser cela, une interface TUN est généralement créée sur chaque point du tunnel. L'interface TUN est associée à un descripteur de fichier permettant la lecture et l'écriture de données. Une fois les interfaces tunnel établies, la copie bidirectionnelle des données entre les interfaces TUN s'effectue à l'aide d'une fonction appropriée.

Lorsqu'un paquet IPv4 est émis depuis le réseau source, il est encapsulé dans un paquet IPv6, qui est ensuite transmis via le réseau IPv6 jusqu'à l'autre extrémité du tunnel. À cet endroit, le paquet est désencapsulé pour récupérer le paquet IPv4 d'origine, qui est ensuite transmis au réseau de destination.

L'objectif de cette partie est de mettre en place ce tunnel.

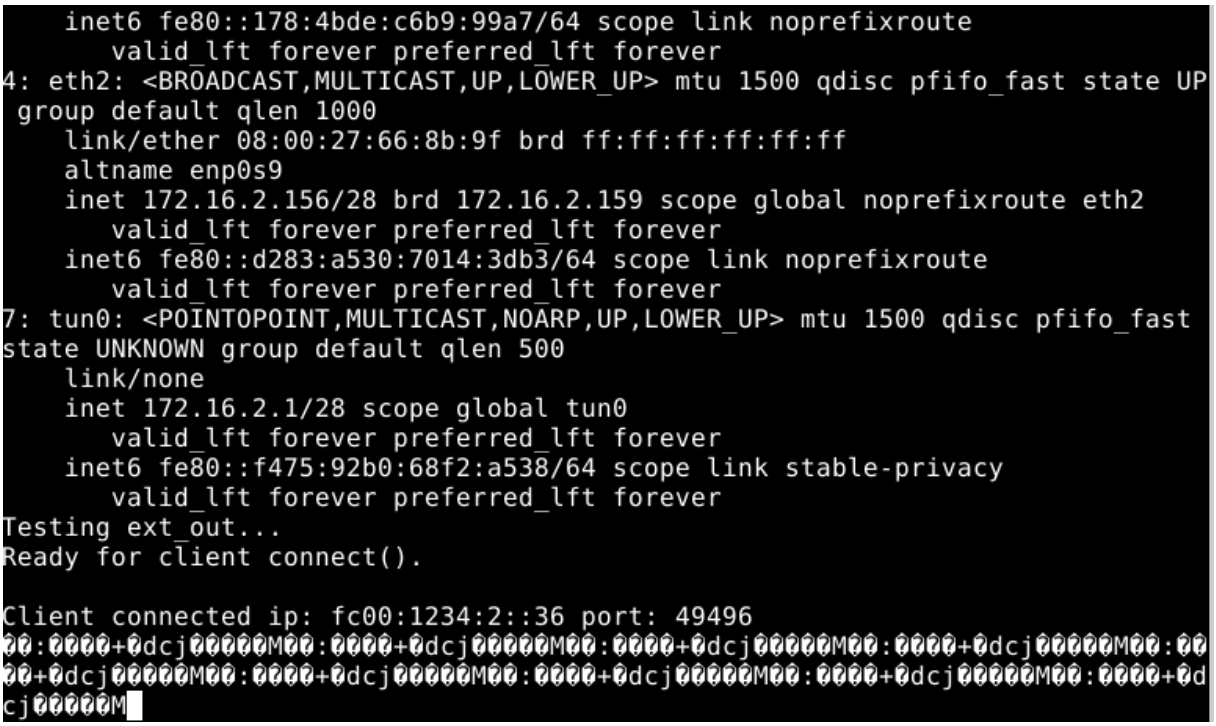
4.1. Redirection du trafic entrant :

Le code vise à établir une bibliothèque appelée "extremite" qui prend en charge le trafic entre les extrémités du tunnel IPv4 sur IPv6. Plus précisément, une fonction nommée "ext-out" est développée pour créer un serveur écoutant sur le port par défaut 123. Cette fonction est conçue pour recevoir les données entrantes sur ce serveur et les rediriger vers la sortie standard. En d'autres termes, elle agit comme une interface entre le réseau IPv6 et la sortie standard, permettant la transmission des données reçues depuis l'extrémité du tunnel IPv6 vers l'application locale.

Parallèlement, une deuxième fonction, "ext-in", est mise en œuvre pour établir une connexion TCP avec l'autre extrémité du tunnel. Cette fonction lit ensuite le trafic provenant de l'interface tun0 et le retransmet à travers la tun0.

La fonction copy() mentionné dans la section précédente est utilisé pour redirigé les données reçues vers la sortie standard (pour ext_out) ou de tun0 vers la socket (pour ext_in).

En mettant en place ext_out dans VM1-6 et ext_in dans VM3-6, on peut voir les données échangées dans le terminal (en injectant du trafic avec ping 172.16.2.10) :



4.2. Redirection du trafic sortant :

Dans cette partie, la fonction `ext-out` de la bibliothèque `extremite` est enrichie pour créer une extrémité capable de rediriger le trafic provenant de la socket TCP vers l'interface tun0 locale. L'initialisation d'un serveur sur le port 123 est maintenue, et lorsqu'une connexion est établie avec l'autre extrémité du tunnel, la fonction entre dans une boucle perpétuelle. À l'intérieur de cette boucle, la fonction utilise la fonction `copier` pour lire continuellement les données de la socket TCP et les réémettre vers l'interface tun0 locale, au lieu de la sortie standard.

Lorsque l'on écrit dans le fichier correspondant au descripteur de l'interface tun0, les données écrites sont injectées dans le réseau en tant que paquets. Ecrire dans l'interface tun0 implique que les données seront encapsulées en paquets IPv6 et transmises à travers le réseau IPv6. Ces paquets, une fois arrivés à l'extrémité de destination du tunnel, seront désencapsulés pour récupérer les données IPv4 d'origine.

VM1-6 :

VM3-6 :

En d'autres termes, écrire dans l'interface tun0 signifie injecter des données dans le tunnel, et ces données seront ensuite acheminées à travers le réseau IPv6 pour atteindre l'autre extrémité du tunnel. C'est ainsi que le trafic est transféré entre les réseaux IPv4 distants en utilisant le réseau IPv6 comme moyen de transport.

4.3. Intégration finale du tunnel :

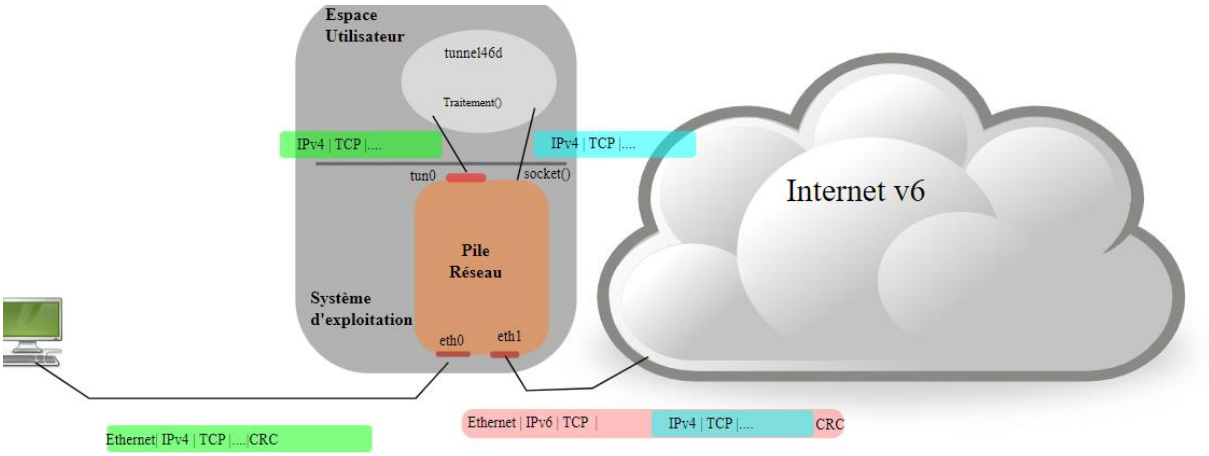
Pour permettre un flux bidirectionnel dans la bibliothèque, il suffit tout simplement de copier le trafic dans les deux sens.

Pour ce faire, une nouvelle fonction est proposée : `bidirectional_tunnel` implémente un tunnel bidirectionnel entre deux descripteurs de fichiers, `fd1` et `fd2`. Ces descripteurs de fichiers représentent une interface TUN (`tun0`) et une connexion TCP établie avec l'autre extrémité du tunnel (une socket). La boucle principale de la fonction assure une copie continue des données entre `fd1` et `fd2`, garantissant ainsi une communication bidirectionnelle asynchrone.

À chaque itération de la boucle, la fonction utilise la fonction `copy_tun` pour copier les données de `fd1` vers `fd2`. Ensuite, elle effectue une copie des données de `fd2` vers `fd1`. Cette approche bidirectionnelle permet à la fonction de traiter les données provenant des deux côtés du tunnel de manière simultanée.

Les communications sont rendues asynchrone grâce à l'utilisation de la fonction `poll`. La fonction `poll` permet de surveiller plusieurs descripteurs de fichiers en même temps et de détecter ceux qui sont prêts pour des opérations d'entrée/sortie sans bloquer le programme. La fonction `poll` est configurée pour attendre des événements de lecture (`POLLIN`). Lorsqu'un des descripteurs de fichiers est prêt pour la lecture, la boucle exécute l'opération de copie (`copy_tun`) de manière non bloquante.

4.4. Mise en place du tunnel entre VM1-6 et VM3-6 : Schémas :



Dans ce cas, VM1 envoie un ping vers VM3 : comme il n'existe pas de route directe, les paquets passeront par notre tunnel.

Les paquets seront encapsulés au niveau de VM1-6, puis passeront par le chemain VM1-6, VM2-6, VM3-6, pour etre décapsulé au niveau de VM3-6 et envoyés à VM3.

4.5. Mise en place du tunnel entre VM1 et VM3 : Système :

Dans cette partie, un fichier 'tunnel46d' est créer afin de mettre en place le système, en lisant tout simplement des fichiers conf.txt propre à chaque machine.

Ces fichiers sont joints avec ce rapport, notamment 'conf_VM1-6' pour VM1-6 et 'conf_VM3-6' pour VM3-6.

5. Validation fonctionnelle :

Vérifications des routes sur les machines :

VM1 :

```
                                vm1

 m1reseaux@VM1:/mnt/partage$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
:
:
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:c9:9c:33 brd ff:ff:ff:ff:ff:ff
    altname enp0s8
    inet 172.16.2.151/28 brd 172.16.2.159 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::dce6:ee51:5a60:10d9/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

m1reseaux@VM1:/mnt/partage$ ip r
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
172.16.2.0/28 via 172.16.2.156 dev eth1 proto static metric 100
172.16.2.144/28 dev eth1 proto kernel scope link src 172.16.2.151 metric 100
172.16.2.176/28 via 172.16.2.156 dev eth1 proto static metric 100
```

VM1-6 :

```
                                VM1-6
 m1reseaux@VM1-6:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
:
:
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:9a:2f:0b brd ff:ff:ff:ff:ff:ff
    altname enp0s8
    inet6 fc00:1234:1::16/64 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::32d:cbb2:40f1:bd07/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:1f:55:c3 brd ff:ff:ff:ff:ff:ff
    altname enp0s9
    inet 172.16.2.156/28 brd 172.16.2.159 scope global noprefixroute eth2
       valid_lft forever preferred_lft forever
    inet6 fe80::d6d3:ad00:3590:2de9/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
7: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 172.16.2.1/28 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 fe80::a78f:4468:6c1e:1adc/64 scope link stable-privacy
       valid_lft forever preferred_lft forever

m1reseaux@VM1-6:~$ ip r
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
172.16.2.0/28 dev tun0 proto kernel scope link src 172.16.2.1
172.16.2.144/28 dev eth2 proto kernel scope link src 172.16.2.156 metric 101
172.16.2.176 via 172.16.2.10 dev tun0
```

VM2-6 :

```
                              VM2-6

  m1reseaux@VM2-6:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
:
:
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:b6:ab:47 brd ff:ff:ff:ff:ff:ff
    altname enp0s8
    inet6 fc00:1234:1::26/64 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::e946:2230:d670:b8cc/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:d7:96:e2 brd ff:ff:ff:ff:ff:ff
    altname enp0s9
    inet6 fc00:1234:2::26/64 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::c379:b32d:6d09:b697/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

m1reseaux@VM2-6:~$ ip r
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
```

VM3-6 :

```
                          VM3-6

m1reseaux@VM3-6:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
:
:
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:d9:cd:7b brd ff:ff:ff:ff:ff:ff
    altname enp0s8
    inet6 fc00:1234:2::36/64 scope global noprefixroute
       valid_lft forever preferred_lft forever
    inet6 fe80::7a44:1735:c8c0:e1b3/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:26:36:af brd ff:ff:ff:ff:ff:ff
    altname enp0s9
    inet 172.16.2.186/28 brd 172.16.2.191 scope global noprefixroute eth2
       valid_lft forever preferred_lft forever
    inet6 fe80::b56:44ae:1e40:b51e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
7: tun0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group default qlen 500
    link/none
    inet 172.16.2.1/28 scope global tun0
       valid_lft forever preferred_lft forever
    inet6 fe80::a0d1:cf9:3cf8:e731/64 scope link stable-privacy
       valid_lft forever preferred_lft forever

m1reseaux@VM3-6:~$ ip r
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
172.16.2.0/28 dev tun0 proto kernel scope link src 172.16.2.1
172.16.2.144 via 172.16.2.10 dev tun0
172.16.2.176/28 dev eth2 proto kernel scope link src 172.16.2.186 metric 101
m1reseaux@VM3-6:~$
```

VM3 :

```
                VM3

m1reseaux@VM3:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
:
:
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:15:cd:83 brd ff:ff:ff:ff:ff:ff
    altname enp0s8
    inet 172.16.2.183/28 brd 172.16.2.191 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::e673:6da4:f983:887e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

m1reseaux@VM3:~$ ip r
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
172.16.2.0/28 via 172.16.2.186 dev eth1 proto static metric 100
172.16.2.144/28 via 172.16.2.186 dev eth1 proto static metric 100
172.16.2.176/28 dev eth1 proto kernel scope link src 172.16.2.183 metric 100
```

Ping VM1, VM3 :

```
m1reseaux@VM1:~$ ping 172.16.2.183
PING 172.16.2.183 (172.16.2.183) 56(84) bytes of data.
From 172.16.2.156 icmp_seq=1 Destination Net Unreachable
From 172.16.2.156 icmp_seq=2 Destination Net Unreachable
From 172.16.2.156 icmp_seq=3 Destination Net Unreachable
From 172.16.2.156 icmp_seq=4 Destination Net Unreachable
^C
--- 172.16.2.183 ping statistics ---
13 packets transmitted, 0 received, +4 errors, 100% packet loss, time 12265ms
```

6. Améliorations :

Par manque de temps, nous n'avons contribué à aucune amélioration parmi les proposées.