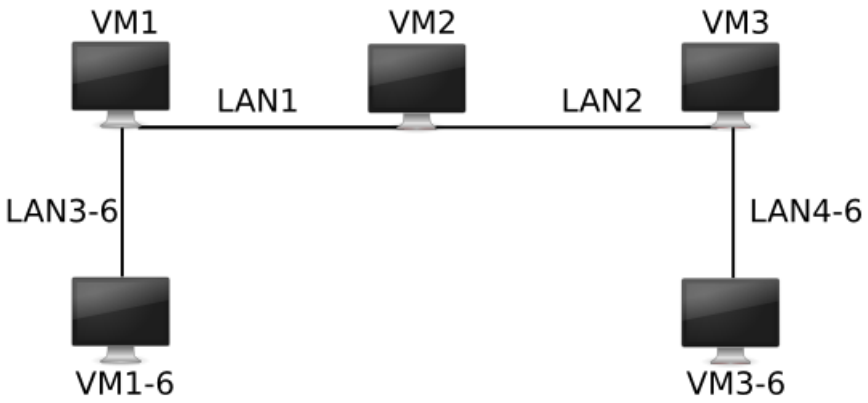
Tunnel IPv6 sur IPv4

M1 Informatique

SCHNEEBERGER Thibault, CHAPUT Jean

Table des matières

1 Configuration réseau



Pour démarrer, on s'intéresse à la réalisation du réseau de 5 machines proposé dans l'énoncé du sujet dont le schéma se trouve juste au dessus. Pour cela, on reprend notre configuration du TP précédent en y apportant quelques modifications. Premièrement, on supprime la configuration de la deuxième interface dans les fichiers de configuration ansible sur les machines `VM1-6` et `VM3-6` car ces machines n'en auront plus la nécessité. (Cela n'est pas nécessaire pour le bon fonctionnement mais ça permet de faire un peu de propre). On supprime de plus les routes devenues obsolètes permettant auparavant le passage par `VM2-6` pour la communication en `IPv6`. Il s'agit là d'une étape obligatoire pour éviter une perte de paquet par la suite.

2 Interface virtuelle

Une fois notre configuration terminée avec ansible, on s'intéresse maintenant à la création et à la configuration d'une interface virtuelle `TUN` qui nous permettra d'effectuer la communication entre l'espace noyau (d'où provient une trame échangée sur le réseau) vers l'espace utilisateur (où se trouvera le code de notre tunnel).

2.1 Création de l'interface

Afin de créer l'interface virtuelle `TUN`, on récupère le code fourni contenu dans le fichier `tunalloc.c`. Celui-ci contient la fonction `tun_alloc()` et une fonction principale `main()`. Lorsque l'on regarde d'un peu plus près la fonction `tun_alloc()`, on remarque que lors de son appel elle retourne un entier. Il s'agit du descripteur de fichier permettant la lecture ou bien l'écriture sur cette interface.

On crée donc une bibliothèque `iftun`, c'est-à-dire un fichier d'en-tête `iftun.h` et un fichier source `iftun.c` afin d'y ajouter la fonction `tun_alloc()`. En plus de cela on crée un fichier principal avec une fonction `main()` nous permettant d'effectuer les tests de cette partie et les suivants.

2.2 Configuration de l'interface

Pour configurer l'interface TUN, on a besoin de lui attribuer une addresse IPv6. Nous utiliseront l'adresse fc00:1234:ffff::1 avec le masque en /32. On choisit un masque en /32 car on remarque qu'à la création de tun0 une route est ajoutée vers fc00:1234::/32. Cela permettra le routage des paquets entrants en direction de LAN4-6 ou LAN3-6 via l'interface tun0 et donc le tunnel. Comme rappelé un peu plus haut il est aussi nécessaire de modifier les configurations créées au TP précédent car dans notre cas certaines routes ne sont plus valides. Par exemple pour VM1 et VM1-6, dans le cas où ces machines voudraient communiquer avec VM3-6, elles ne pourront plus passer par VM2-6.

On crée donc un script configure_tun.sh qui contiendra les commandes de configuration de l'interface virtuelle suivantes. ip address add fc00:1234:ffff::1/32 dev tun0 pour lui ajouter l'adresse IP. Mais aussi la commande ip link set dev tun0 up afin d'activer l'interface qui était pour le moment en état down après sa création.

Afin de tester notre configuration et après avoir lancé notre programme de test sur VM1, on tente d'effectuer un ping -6 de VM1-6 vers l'interface tun0 sur VM1. Celui-ci se déroule sans problème et reçoit une réponse. Cependant, si l'on effectue une capture wireshark sur VM1, on se rend compte que les échanges ne passent pas par tun0. En effet, celle-ci ne capte aucune trame. Cela peut s'expliquer car lorsque la trame envoyée par VM1-6 arrive sur VM1 par son interface eth2 (passerelle par défaut de VM1-6) elle est désencapsulée afin de regarder qui est le destinataire pour pouvoir lui transmettre dans une nouvelle trame. Voyant que le destinataire est en réalité lui-même, le routeur VM1 peut traiter la demande et envoyer sa réponse. Cela se déroule dans l'espace noyau.

On effectue ensuite le test en faisant un ping -6 depuis VM1 vers l'addresse IPv6 suivante : fc00:1234:ffff::10. Cette fois-ci le ping ne reçoit pas de réponse mais en effectuant une analyse de paquets avec wireshark sur VM1, on se rend compte que les paquets sont transmis depuis l'interface eth2 de VM1 vers tun0 .Le ping ne reçoit ainsi pas de réponse car pour le moment les paquets injectés dans tun0 ne sont pas traités. Attention car pour cette partie il faut bien penser à ajouter le drapeau de routage IPv6 sur VM1 (et par symétrie sur VM3) sinon la transmission entre interfaces d'une même machine ne pourra pas s'effectuer.

2.3 Récupération des paquets

Maintenant que l'on sait que les paquets sont redirigés sur l'interface virtuelle tun0 dont nous disposons du descripteur de fichier, on peut s'intéresser à la récupération des informations depuis tun0. On complète donc notre bibliothèque iftun avec une nouvelle fonction transfert qui permettra de transférer les données lues sur un descripteur de fichier source vers un descripteur de fichier destination.

On effectue donc le test de cette fonction avec comme descripteur de fichier source celui retourné par la création de l'interface tun0 et comme descripteur de fichier destination 1 (qui correspond à la sortie standard). On effectue à nouveau les pings réalisés précédemment depuis VM1-6. Au niveau du réseau rien ne change, Les captures sont identiques. Cependant, on observe les paquets entrants sur tun0 s'afficher dans le terminal. Ceux-ci étant en binaire, on

les rends plus lisibles en filtrant l'affichage avec `hexdump` afin de rendre cela plus lisible.

Différentes options sont disponibles lorsque l'on crée l'interface virtuelle. On peut utiliser les flags suivants : `IFF_TUN`, `IFF_TAP` et `IFF_NO_PI`. `IFF_TUN` est celui que l'on utilise car il permet de supprimer les en-têtes Ethernet afin de ne garder que le datagramme IP. `IFF_NO_PI` permet de retirer les 4 octets concernant la version du protocole de couche IP. Le flag `IFF_NO_PI` peut s'ajouter en plus du flag `IFF_TUN`, les flags sont cumulables.

3 Un tunnel simple pour IPv6

Afin de faire communiquer `VM1` et `VM3` qui sont reliés par le biais de `VM2` en `IPv4`, on utilisera la connexion par socket à l'image d'un client et d'un serveur. Pour cela on crée une nouvelle bibliothèque `extremite` qui contiendra les fonctions nécessaires à l'envoi et à la réception des datagrammes.

3.1 Redirection du traffic entrant

Afin de rediriger le traffic lu depuis `tun0` vers notre réseau `IPv4`, il nous faut créer une fonction `ext_in()` qui aura en réalité un rôle de client et qui sera chargée d'ouvrir une connexion avec l'autre extrémité du tunnel afin de lui envoyer tout ce qui est lu. Après avoir ouvert une connexion distante sur sa socket, on peut simplement effectuer un appel à notre fonction `transfert()` de la bibliothèque `iftun`. Celle-ci prend deux descripteurs de fichiers en paramètres, le premier étant celui retourné par `tun0` et le deuxième la socket en elle-même.

En plus de la fonction `ext_in()`, il nous faut aussi une fonction `ext_out()` qui sera chargée du rôle de serveur et qui écoutera donc sur sa socket afin de rediriger dans un premier temps vers la sortie standard (descripteur de fichier 1) afin de vérifier que tout fonctionne correctement. La redirection est ici aussi effectuée à l'aide de la fonction transfert de la bibliothèque `iftun`. À terme le transfert ne s'effectuera plus vers la sortie standard mais bien sur le descripteur de fichier de `tun0` afin de réinjecter notre datagramme `IPv6` dans le réseau.

Lorque tout est mis en place, c'est à dire que le programme de test est lancé en mode client sur `VM1` et en mode serveur sur `VM3`, on peut effectuer notre `ping -6` habituel depuis `VM1-6` vers l'adresse `fc00:1234:ffff::10`. Si l'on se place ensuite sur `VM3`, on voit l'ensemble des paquets arriver depuis le tunnel s'afficher sur la sortie standard.

3.2 Redirection du traffic sortant

Maintenant qu'il est possible pour nous d'afficher le contenu du tunnel sur la sortie standard de `VM3`, on veut pouvoir le rediriger vers son interface virtuelle `tun0`. Cela aura pour but de réintégrer le datagramme `IPv6` transporté par le tunnel sur `IPv4` dans le réseau `IPv6`. Pour cela il suffit de modifier la fonction `ext_out()` de la bibliothèque `extremite` en effectuant un transfert non plus vers 1 (sortie standard) mais vers le descripteur de fichier retourné par `tun0`.

De cette manière les datagrammes `IPv6` qui étaient transportés par un datagramme `IPv4` dans le tunnel sont réintégrés dans un réseau `IPv6` à la sortie de celui-ci. Cela permet à la

machine VM3-6 de recevoir le ping provenant de VM1-6. Pour vérifier le bon fonctionnement du tunnel, on peut prendre un datagramme IPv6 avec comme adresse de destination la machine VM3-6 et l'envoyer sur la socket d'écoute de la sortie du tunnel sur VM3. On réalise cela à l'aide de la commande `nc 172.16.2.163 123 < datagramme-v6.bin`. On se place sur VM3 et on observe le paquet entrant être redirigé vers tun0 puis arriver sur VM3-6.

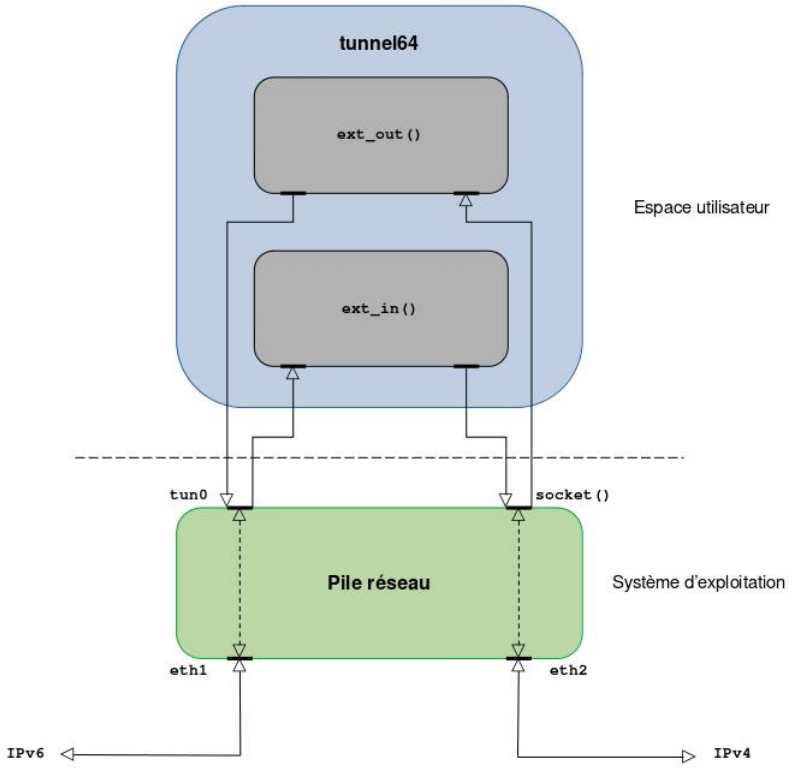
3.3 Intégration finale du tunnel

Maintenant que notre datagramme IPv6 arrive à rejoindre notre second réseau IPv6, on cherche à faire fonctionner notre tunnel dans les sens. Pour cela, il nous faut lancer les fonctions `ext_in()` et `ext_out()` à chaque extrémité du tunnel. Afin que cela soit possible on décide de passer l'adresse IPv4 de la sortie du tunnel en paramètres de la fonction `ext_in()`. Avec une modification de notre fonction principale `main()`, on lui permet de lancer le tunnel soit sur VM1 soit sur VM2.

Pour lancer à la fois la fonction `ext_in()` et la fonction `ext_out()`, on utilisera la librairie C `pthreads` qui permet de lancer des threads d'exécution assez simplement au lieu d'utiliser les processus système avec des appels à `fork()`. Pour pouvoir lancer ces threads depuis le main, on aura besoin d'une bibliothèque supplémentaire que l'on nommera `executeur`. Cette bibliothèque contient deux pointeurs de fonctions qui font appel à `ext_in()` et `ext_out()`. On crée aussi une structure de donnée permettant de passer les arguments nécessaires au lancement du tunnel à nos fonctions.

Après avoir lancé le tunnel sur VM1 et sur VM2, on effectue des pings entre VM1-6 et VM3-6. Pour vérifier que tout est bon on effectue une capture wireshark du traffic circulant sur VM2 en IPv4. On effectue une seconde capture sur VM3 en IPv6. D'après l'analyse de ces captures situées dans le répertoire docs, tout est fonctionnel.

3.4 Mise en place du tunnel entre VM1 et VM3 : Schémas



Voici le schéma plus détaillé du fonctionnement du tunnel. Lorsqu'un datagramme `IPv6` est à destination de l'autre extrémité du tunnel, il est lu sur l'interface virtuelle `TUN` (`tun0`) et encapsulé dans un datagramme `IPv4` en étant écrit sur la socket ouverte par la fonction `ext_in()`. Il est ensuite acheminé jusqu'à sa destination (c'est-à-dire `VM3` dans notre cas) en empruntant le réseau `IPv4` (`LAN1` et `LAN2` ici). A son arrivée à la sortie du tunnel, le datagramme `IPv6` est désencapsulé du datagramme `IPv4` puis écrit dans l'interface virtuelle `TUN` (`tun0`) afin de rejoindre sa destination en `IPv6`.

3.5 Mise en place du tunnel entre VM1 et VM3 : Système

Pour n'avoir qu'un simple exécutable permettant de démarrer le tunnel, on s'intéresse à la lecture des paramètres de celui-ci depuis un fichier de configuration `tunnel64.conf`. On crée alors une bibliothèque de fonctions `config` dont le rôle est de récupérer les paramètres de configuration dans le fichier afin de pouvoir les faire passer en argument de nos appels à `ext_in()` et `ext_out()`. Une fois cela fait, on peut essayer de démarrer le tunnel sur `VM1` et `VM3` avec la commande `./bin/tunnel64` depuis le répertoire `/mnt/partage/tunnel64`. Le tunnel se lance correctementet les pings entre `VM1-6` et `VM3-6` sont fonctionnels.

4 Validation fonctionnelle

4.1 Configuration

Sur chaque machine de notre réseau (`VM1`, `VM2`, `VM2`, `VM1-6` et `VM3-6`) on récupère les paramètres réseaux significatifs avec les commandes `ip a` et `ip r`. L'ensemble des résultats est cohérent avec ce que nous attendions. Si besoin est de les consulter, l'ensemble des fichiers texte contenant les résultats des commandes se trouvent dans le répertoire `docs/configs`. Les résultats sont aussi disponibles en simples captures d'écran dans le répertoire `docs/images/configs`.

4.2 Couche 3

On se place sur `VM1-6` et on effectue un ping vers `fc00:1234:4::36`. Celui-ci se termine sans erreur et nous donne les captures d'écran situées dans le répertoire `docs/images`. On effectue en plus deux captures à l'aide de wireshark. Une en `IPv4` sur `VM2` et une en `IPv6` sur `VM3`. Ces captures sont aussi disponibles dans le répertoire `docs`. On y voit notre datagramme `IPv6` envoyé par `VM1-6` circuler en données d'un datagramme `IPv4` sur `VM2` puis on le retrouve sur le réseau `IPv6` de `VM3` lorsqu'il est réintroduit sur `tun0`.

4.3 Couche 4

Pour tester un peu plus en profondeur le fonctionnement du tunnel, on essaye de se connecter au service `echo` de `VM3-6` depuis `VM1-6`. Pour cela on utilise la commande telnet suivante : `telnet fc00:1234:4::36 echo`. La connexion se déroule sans problème et le service `echo` fonctionne correctement.

4.4 Couche 4 : bande passante

On s'intéresse maintenant à la performance de notre tunnel. Pour cela, nous allons utiliser l'utilitaire `iperf3`. On commence par le lancer en mode serveur sur `VM3-6` puis on essaye avec différentes tailles de tampon depuis `VM1-6`.

Peu importe la taille de tampon envoyé avec `iperf3`, le débit est en moyenne au dessus d'1Gb/s. Il nous est cependant impossible de déduire une tendance en fonction de la taille du tampon en entrée car les valeurs varient énormément sans corrélation avec celle-ci.

5 Améliorations

5.1 Configuration via ansible

Puisque notre tunnel peut-être lancé simplement depuis le terminal, il semble tout à fait possible de le lancer avec `ansible`. Cependant pour pouvoir faire cela, il nous faut pouvoir lancer le tunnel en tâche de fond afin que celui-ci ne soit pas interrompu lors de la fin de la configuration et à la fermeture du shell. Sur un système `GNU/Linux`, on peut utiliser la commande `nohup` qui permet de faire cela.

Lorsque l'on utilise cette commande dans `ansible`, on est confronté à des erreurs qui sont générées par la sortie standard et la sortie d'erreur. Pour les éviter on redirige la sortie d'erreur vers la sortie standard et la sortie standard vers `/dev/null`.

Une fois tout cela fait, lorsqu'on lance la configuration de `VM1` et `VM3` via ansible avec la commande `ansible-playbook /vagrant/config.yml`, le tunnel se lance. On peut le vérifier en faisant `ip a` sur l'une des deux extrémités du tunnel, `tun0` est encore affichée, ce qui signifie que le tunnel tourne encore. Pour vérifier qu'il fonctionne correctement on effectue à nouveau un ping depuis `VM1-6` vers `VM3-6` avec succès.