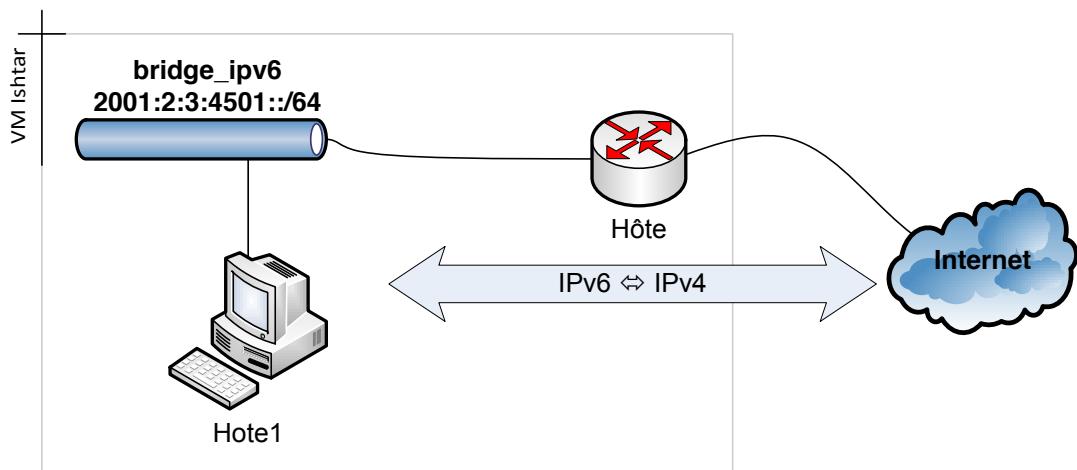


Réalisation d'un traducteur IPv6 ⇔ IPv4 pour le protocole TCP

■ ■ ■ Présentation du projet

Le but du projet est de permettre à une machine n'utilisant que la pile TCP/IPv6 de communiquer avec des machines en IPv4 de manière transparente au travers du protocole TCP :



Le réseau est simulé de la manière suivante :

- un réseau local sous IPv6 possédant un préfixe global, ici `2001:2:3:4501::/64` ;
- un hôte connecté à ce réseau, ici le « netns » « `Hôte1` » sert de machine cliente ;
- un routeur, ici « l'hôte », c-à-d la VM, « `Virtual Machine` », ou votre machine sous Linux, donne l'accès à Internet.

La traduction de protocole IPv6 ⇔ IPv4 se déroule de la manière suivante :

- ▷ la machine cliente obtient le préfixe réseau IPv6 global depuis le routeur :
 - le routeur diffuse un message icmpv6 de type « `router advertisement` » sur le réseau local ;
 - toutes les machines connectées au réseau local :
 - ★ reçoivent le préfixe réseau et définissent leur propre adresse IPv6 globale ;
 - ★ désignent le routeur comme « route par défaut » ;
- ▷ la machine cliente se connecte sur un serveur externe au réseau, suivant son adresse IPv6 globale ;
- ▷ la connexion de la machine cliente est prise en charge par le routeur de la manière suivante :
 - paquet reçu en IPv6 contenant du protocole TCP à destination d'un serveur externe au réseau :
 - ★ désencapsulation du segment TCP ;
 - ★ encapsulation du segment dans un datagramme IPv4 ;
 - ★ envoi du paquet vers le serveur externe **suivant son adresse IPv4** ;
 - paquet reçu en IPv4 contenant du protocole TCP à destination de la machine cliente :
 - ★ désencapsulation du segment TCP ;
 - ★ encapsulation du segment dans un datagramme IPv6 ;
 - ★ envoi du paquet vers la machine cliente ;
- ▷ pour la découverte de l'adresse IPv4 associée à l'adresse IPv6 du serveur externe :
 - interception de la requête DNS de résolution « `adresse symbolique` » ⇒ adresse IPv6, avec le champs AAAA ;
 - réalisation d'une requête DNS de résolution « `adresse symbolique` » ⇒ adresse IPv4 avec le champs A ;
 - sauvegarde de la correspondance (@IPv6, @IPv4) ;
 - utilisation de cette association lors de la demande de connexion de la machine cliente.



■■■■■ Outils à utiliser

Pour ce projet, on utilisera les outils suivants :

- **NetFilter**, le firewall intégré à Linux, en particulier la table « mangle » qui va nous permettre d'intercepter les paquets à l'entrée de la pile TCP/IP grâce à sa chaîne « PREROUTING » ;
- **NFQueue**, la passerelle NetFilter ⇔ « User space » qui va nous permettre de récupérer les paquets IPv4 et IPv6 au sein d'un programme utilisateur en provenance d'une règle de NetFilter ;
- **Scapy** pour l'analyse, la modification, la création et l'injection de paquet IPv4 ou IPv6 ;
- **Radvd** le démon réalisant la diffusion de « *router advertisement* » pour la diffusion du préfixe réseau IPv6 et du routeur prenant en charge le trafic, vers et depuis, l'extérieur du réseau local ;
- **Socat**, comme outil permettant de faire des connexions TCP que ce soit en IPv4 ou en IPv6.

■■■■■ Utilisation de NFQueue & Scapy

Voici le code pour l'utilisation de NFqueue et Scapy dans le même programme Python :

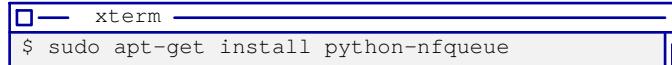
```
#!/usr/bin/python
# coding=utf-8
import nfqueue, socket
from scapy.all import *

def traite_paquet(payload):
    # le paquet est fourni sous forme d'une séquence d'octet, il faut l'importer
    data = payload.get_data()
    # il faut identifier sa nature IPv6 ou IPv4
    premier_quartet = data[0].encode("hex")[0]
    if (premier_quartet == '4') :
        # paquet IPv4
        pkt = IP(data)
    else:
        # paquet IPv6
        pkt = IPv6(data)
    pkt.show()
    # accepte le paquet : le paquet est remis dans la pile TCP/IP et poursuit sa route
    #payload.set_verdict(nfqueue.NF_ACCEPT)
    # si modifie : le paquet est remis MODIFIE dans la pile TCP/IP et poursuit sa
    #route
    #payload.set_verdict_modified(nfqueue.NF_ACCEPT, bytes(pkt), len(pkt))
    # si rejete : le paquet est rejeté
    #payload.set_verdict(nfqueue.NF_DROP)

q = nfqueue.queue()
q.open()
q.unbind(socket.AF_INET)
q.unbind(socket.AF_INET)
q.bind(socket.AF_INET6)
q.bind(socket.AF_INET)
q.set_callback(traite_paquet)
q.create_queue(0)
try:
    q.try_run()
except KeyboardInterrupt, e:
    print "interruption"
q.unbind(socket.AF_INET)
q.unbind(socket.AF_INET6)
q.close()
```

Pour l'utilisation de NFQueue dans notre projet :

- ▷ Pour l'installer sous Ubuntu :



- ▷ **Remarque importante** : NFQueue ne récupère que des datagrammes, jamais des trames Ethernet ;
- ▷ la sélection précise du trafic (protocole TCP, port utilisé, *etc.*) est faite en amont par NetFilter ;
- ▷ la récupération des paquets doit être configurée pour IPv6 et IPv4 (deux « AF_family ») ;
- ▷ le datagramme récupéré est soit IPv6, soit IPv4 : analyse du premier quartet, « *nibble* », du paquet ;
- ▷ la méthode « *set_verdict* » permet de décider du traitement par NetFilter : « acceptation tel quel », « acceptation après modification » ou bien « rejet » ;

- ▷ pour la décision à prendre quant à l'avenir du datagramme, deux possibilités :
 - ◊ `payload.set_verdict_modified(nfqueue.NF_ACCEPT...)` : on pourrait l'utiliser après avoir traduit le paquet IPv6 vers IPv4 ou vice-versa :
 - * **problème** : le paquet reste dans la pile TCP/IP d'origine (IPv6 ou IPv4) et, s'il est redirigé vers une interface externe, il est encapsulé dans une trame avec un « ethertype » correspondant à sa nature d'origine ! (0x800 pour de l'IPv6 et de 0x86DD pour de l'IPv4, ce qui rendra son interprétation lors de sa réception incorrecte).
- Vous pouvez essayer pour voir l'analyse que va faire `tcpdump` !
- ◊ `payload.set_verdict(nfqueue.NF_DROP)` permet de faire disparaître le datagramme sans qu'il ne rentre dans la pile TCP/IP (v6 ou v4) du routeur :
- * on se servira de Scapy pour faire l'envoi dans une trame de nature correcte (et avec les bonnes adresses MAC source et destination) :

```
# pour envoyer un datagramme IP sur l'interface eth0
send(pkt, iface="eth0")
# pour envoyer une trame ethernet sur l'interface bridge_ipv6
sendp(pkt, iface="bridge_ipv6")
```

■■■ Utilisation de NetFilter en combinaison avec NFQueue

Pour l'utilisation de NetFilter dans notre projet :

- ▷ nous utiliserons la table « mangle » et la chaîne « PREROUTING » afin d'intercepter le datagramme avant sa prise en charge par la pile TCP/IP ;
- ▷ nous sélectionnerons le protocole TCP et un port destination donné (par ex. le port 7890) afin de tester notre traducteur ;
- ▷ nous indiquerons deux règles pour rediriger le trafic vers la queue N°0 de NFQueue :
 - ◊ en IPv4 :

```
xterm
sudo iptables -t mangle -A PREROUTING -p tcp --sport 7890
-j NFQUEUE --queue-num 0
```

- ◊ en IPv6 :

```
xterm
sudo ip6tables -t mangle -A PREROUTING -i bridge_ipv6 -p tcp ---dport 7890
-j NFQUEUE ---queue-num 0
```

Vous vérifieriez que le trafic en IPv4 va du serveur TCP vers le client, et que le trafic en IPv6 va du client vers le serveur TCP.

■■■ Utilisation de radvd

Pour l'installer, s'il ne l'est pas déjà :

```
xterm
$ sudo apt-get install radvd
```

Pour l'utiliser, on lancera le script suivant :

```
#!/bin/bash

INTERFACE=switchipv6
CONFIG=$(cat <<END
interface $INTERFACE
{
    AdvSendAdvert on;
    MinRtrAdvInterval 5;
    MaxRtrAdvInterval 15;
    prefix 2001:2:3:4501::/64
    {
        AdvOnLink on;
        AdvAutonomous on;
    };
};
END
)

radvd -C <(echo "$CONFIG")
```

avec :

```
xterm
pef@cube:~/IPV6$ sudo ./script_radvd
```

Vous pourrez vérifier qu'il fonctionne :

```
xfce4-terminal -x xterm
pef@cube:~/IPV6$ sudo tcpdump -i switchipv6 -lnvv ip6
12:33:47.340379 IP6 (flowlabel 0x65aa4, hlim 255, next-header ICMPv6 (58) payload length: 56)
fe80::62da:96a2:93b9:1a05 > ff02::1: [icmp6 sum ok] ICMP6, router advertisement, length 56
    hop limit 64, Flags [none], pref medium, router lifetime 45s, reachable time 0s, retrans time 0s
    prefix info option (3), length 32 (4): 2001:2:3:4501::/64, Flags [onlink, auto], valid time 86400s, pref. time 14400s
        0x0000: 40c0 0001 5180 0000 3840 0000 0000 2001
        0x0010: 0002 0003 4501 0000 0000 0000 0000
    source link-address option (1), length 8 (1): 78:92:9c:e4:5b:83
        0x0000: 7892 9ce4 5b83
```

et :

```
xfce4-terminal -x xterm
pef@cube:~/IPV6$ ip address show switchipv6
2: wlp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 78:92:9c:e4:5b:83 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.130/24 brd 192.168.0.255 scope global wlp1s0
        valid_lft forever preferred_lft forever
    inet6 2001:2:3:4501:8a2:c3d9:697f:c169/64 scope global temporary dynamic
        valid_lft 86400sec preferred_lft 14400sec
    inet6 2001:2:3:4501:48a7:631f:b007:3e71/64 scope global mngtmpaddr noprefixroute dynamic
        valid_lft 86400sec preferred_lft 14400sec
    inet6 2001:41d0:fe2e:7aa:f/128 scope global dynamic
        valid_lft 85155sec preferred_lft 85155sec
    inet6 fe80::62da:96a2:93b9:1a05/64 scope link
        valid_lft forever preferred_lft forever
```

Vous vérifierez que la VM et l'hôte se configureront bien une adresse IPv6 à l'aide de ce préfixe.

Sur les dernières versions d'Ubuntu, il peut être nécessaire d'« autoriser » la configuration automatique du poste à la réception d'un « *router advertisement* » :

```
xfce4-terminal -x xterm
sudo sysctl -w net.ipv6.conf.bridge_ipv6.accept_ra=2
```

À exécuter sur la VM comme sur le netns.

Pour l'utilisation de « socat » :

- * en mode serveur en IPv4 :

```
xfce4-terminal -x xterm
$ socat - tcp-listen:7890
```

- * en mode client en IPv6 :

- vers une adresse globale :

```
xfce4-terminal -x xterm
$ socat - tcp6:[2001:41d0:2:eb43::b7f9:7a68]:7890
```

- vers une adresse locale de lien :

```
xfce4-terminal -x xterm
$ socat - tcp6:[fe80::e02b:17ff:fe97:db47%eth0]:7890
```

Pour une adresse de lien, on doit indiquer l'interface à utiliser.

■■■ Quelques indications & remarques

Pour le choix de l'adresse IPv4 d'origine du trafic TCP

Vous utiliserez l'adresse IPv4 de la VM, si vous en utilisez une, ou bien celle de votre machine réelle sous Ubuntu si vous l'utilisez.

Le « routeur » ne fera pas de NAT parce qu'il traduira, grâce à votre programme, le datagramme IPv6 en IPv4, et vice-versa.

L'utilisation du NAT est déconseillé en IPv6 !

Pour l'établissement de la connexion en mode client depuis le container « Host1 », vérifiez l'adresse IPv6 globale utilisée pour cet établissement : sur les versions récentes d'Ubuntu, l'adresse IPv6 utilisée peut être une adresse différente, choisie aléatoirement, de celle résultant de la combinaison de l'adresse MAC et du préfixe global (protection contre le traçage de l'adresse IPv6 dérivée de l'adresse MAC du matériel).

Vérifiez toujours à l'aide de « `tcpdump` » que le trafic TCP que vous manipulez est bien comme vous l'attendez et contenu dans une trame ethernet avec la bonne destination :

- ◊ Scapy a un soucis avec l'exploitation de sa table de routage (le fameux message d'erreur lorsqu'il indique ne pas disposer d'adresse de route par défaut en IPv6) ;
- ◊ vous pouvez afficher la table de routage utilisée par Scapy à l'aide de l'instruction :

```
print conf.route6
```

- ◊ vous pouvez enlever la route par défaut utilisée par Scapy à l'aide de l'instruction suivante :

```
conf.route6.delt(dst=':::/0')
```

- ◊ le problème concerne l'envoi d'une trame par Scapy contenant le datagramme IPv6 forgé : il l'envoie à l'adresse MAC du routeur s'il dispose d'une route par défaut pour IPv6 (destination `::/0`) ou de diffusion IPv4 `ff:ff:ff:ff:ff:ff` s'il n'a pas de route par défaut en IPv6.

Le plus simple dans ces conditions est celle suggérée dans le sujet : vous construisez vous-même votre trame Ethernet pour l'envoi de datagramme IPv6 (en particulier, avec la bonne adresse MAC de destination de la machine cliente ; pour celle de source, Scapy y arrive tout seul).

■■■ Travail demandé

Votre programme devra fonctionner uniquement pour une adresse IPv6 donnée (fausse dans la mesure où nous ne disposons pas de connectivité IPv6 sur l'Université), vers une adresse IPv4 donnée.

- a. Vous écrirez le programme Python réalisant ce « traducteur IPv6 ⇌ IPv4 » en fixant dans le programme l'adresse IPv6 du serveur externe, l'adresse IPv4 du serveur externe, l'adresse IPv6 du client, l'adresse MAC du client.
- b. Vous écrirez un rapport au format PDF avec :
 - ◊ des captures :
 - * de trafic TCP en sortie de la VM (datagrammes IPv4) ;
 - * de trafic TCP sur le « `bridge_ipv6` » (datagrammes IPv6) ;
 - Ces deux captures seront à **commenter** pour valider votre traducteur.
 - ◊ l'explication du déroulement de l'établissement de la connexion TCP au travers du traducteur.
 - ◊ la trace de l'exécution des commandes « `socat` » en client et serveur, avec un échange de messages.

Remise du travail

Le travail devra être remis sous forme d'une archive déposée sur **Communities**.

Cette archive contiendra vos sources (configuration conteneur, routage, programme Python, *etc.*) ainsi que le rapport au format PDF.