Devoir UE9
Réseaux et protocoles

Simulation d'un réseau Ipv4/IPv6

Julien Langlois – Jérôme Groven

Décembre 2005

Table des matières

Introduction

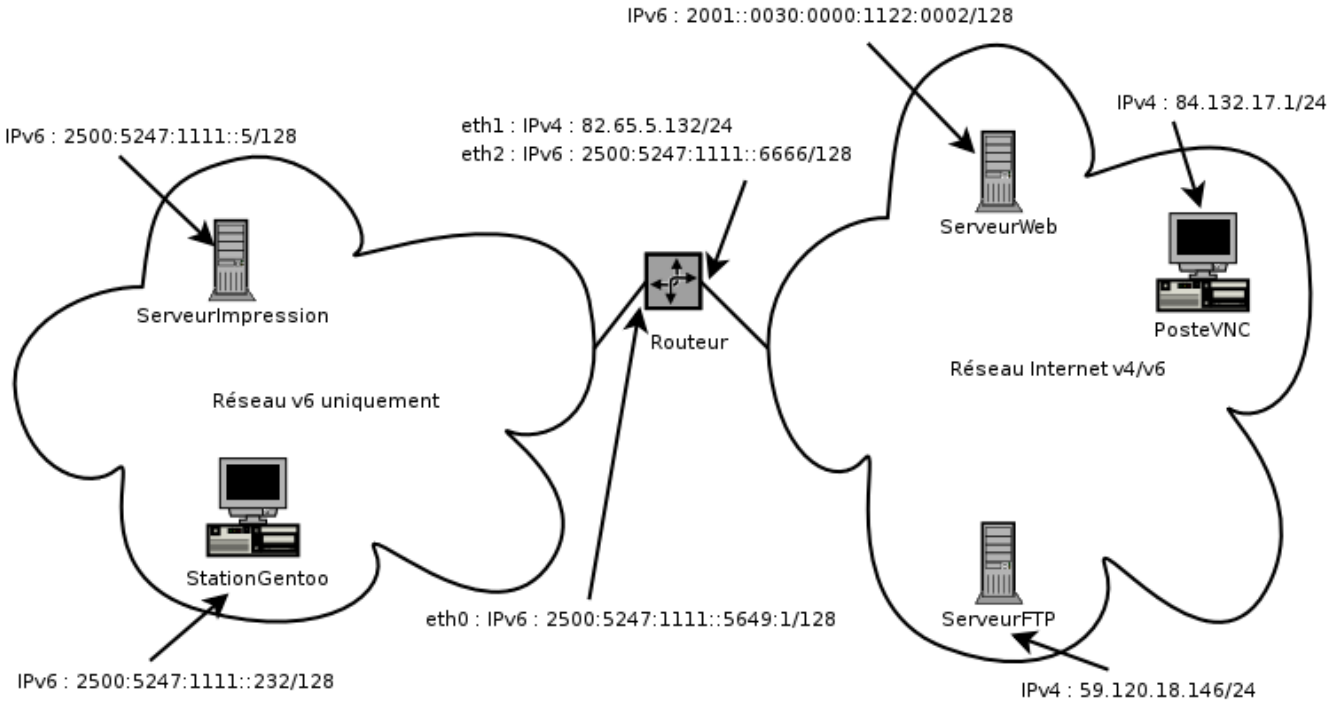
Le but de ce projet est créer une simulation permettant de communiquer entre un réseau interne entièrement v6 et un réseau externe v4/v6. Pour cela, nous avons à disposition la couche de transport UDP qui nous permet de faire différentes requêtes au sein du réseau.

Il est également demandé d'implémenter le fonctionnement d'une passerelle/routeur DNS supportant le mécanisme DNS-ALG pour ce qui est de la communication d'une adresse v6 avec une adresse v4.

Nous allons tâcher d'expliquer de quelle manière nous avons choisi de réaliser ce projet.

I. Configuration de la simulation

Voici le réseau utilisé pour la simulation :



II. Choix du language : Perl

La raison pour laquelle nous avons choisi Perl est simple : nous connaissions très peu ce langage et nous n'avions jamais programmé de gros projets en Perl. Notre connaissance dans ce langage se limitait à quelques scripts.

Après quelques recherches , nous avons trouvé comment marcher la couche objet de Perl : un objet est une simple table de hashage. Cette structure est très souple. De plus, le langage Perl hérite de beaucoup de fonctionnalités du langage C en y apportant une couche de souplesse et de renouveau.

Au vu de ces faits, ce langage nous est apparu être un langage adapté pour ce devoir.

III. Structures utilisées

Nous avons choisi de développer une simulation dynamique permettant à un ou plusieurs utilisateurs d'envoyer et recevoir des paquets sur le réseau. Pour cela, nous avons choisi de faire un système multiprocessus. C'est à dire que chaque machine du réseau est lancée séparement.

Thread

Afin de rester en mode dynamique, il fallait pouvoir avoir un prompt attendant la une chaîne de caractères rentrée par l'utilisateur et, en meme temps, pouvoir lire sur le réseau. Nous avons donc choisi de lancer ce prompt dans un thread. De cette manière, la machine lit le réseau en continu sans être bloquée par l'attente de l'interaction avec l'utilisateur.

Tube nommé

Comme nous étions en multiprocessus, il fallait trouver un moyen de communiquer entre les machines. Notre choix s'est tout d'abord tourné sur de simple de fichier, mais la tache était fastidieuse car il fallait gérer le vérouillage des fichiers pour l'écriture, plus le mode FIFO (l'application réseau doit lire le plus ancien paquet reçu).

C'est alors que nous nous sommes rappellés que l'architecture Unix/Linux proposé ce genre de service : les tubes nommés.

Les tubes nommés vont alors représenter les interfaces réseau (les points d'accroches au réseau).

IV. Fonctionnement du simulateur

Fichiers de configuration machines

Afin de ne pas avoir une application figée dans la configuration de base proposée par le sujet, nous avons choisi de stocker les configurations des différentes machines dans des fichiers de configuration. Cela nous a permis de vérifier le bon fonctionnement des scénarios dans la configurations proposé, mais également de tester sur d'autres configurations avec plus ou moins de machines lancées.

```
### Commentaires
HOST nom_de_machine
INTERFACE Out v4=83.157.0.89
INTERFACE In v6=2001:0000:0000:0000:0000:0000:0000:2013
IPFORWARD
VERBOSE 10
```

Les lignes commençant par des # sont assimilées à des commentaires et ne sont pas interprétées.

Lorsqu'une ligne commence par HOST, le mot d'après représente le nom de machine.

Lorsqu'il s'agit du mot INTERFACE, on assimile le deuxième mot au domaine (réseau) dans laquelle cette interface se connecte. Ensuite, les mot suivants réprésentent les IPs avec leur version (v4=84.157.0.89 => IPv4 de valeur ....).

Le terme IPFORWARD déclenche le mode routeur, c'est à dire le lancement du serveur DNS, les distributions d'adresses DNS-ALG et le forward d'IPv6 a travers les interfaces.

La valeur du VERBOSE va permettre à l'utilisateur d'afficher plus ou moins de messages à l'écran selon la couche réseau qu'il veut observer.

Simulation du réseau

Le réseau est simulé par une ensemble de fichiers et de répertoires.

Tout d'abord, toute la simulation se trouve dans le répertoire “Reseau”. Dans ce répertoire “Réseau” se trouve des sous-répertoires représentant chaque sous-réseau. Par exemple, le réseau Internet est réprésenté par le répertoire “Out”.

Dans un sous répertoire se trouve 2 types de fichiers :

- le fichier ROUTE : la table de routage du sous réseau
- des tubes avec pour nom des adresses IP.

Un tube représente le point d'accroche d'une interface d'une machine à ce réseau.

Connexion

Quand une interface se connecte au réseau, elle commence par créer le tube. Ensuite, elle ouvre ce tube en lecture (un thread lancé précédement ouvre ce tube en écriture pour ne pas bloquer la procédure). Ensuite, l'interface va renseigner la table de routage. Elle va ouvrir le fichier ROUTE et y ajouter son IPv4 et/ou son IPv6 ainsi que son nom d'hôte.

Lorsqu'une machine a l'option IPFORWARD d'activée, elle renseigne la table de routage en y incorporant les lignes “default:” (dans un sous réseau) ou un “DOMAIN:” (dans le réseau Internet) selon les différentes interfaces.

Toutes les machines d'un sous réseau ont donc la même table de routage. Seul un routeur a plusieurs tables de routage.

Envoi d'un paquet

Lorsqu'une machine veut envoyer un paquet, l'application gère la construction du datagramme et ordonne à la partie réseau de l'envoyer.

C'est alors que la partie réseau essaye de l'envoyer sur chacune de ses interfaces. Quand une interface a réussi à envoyer le paquet, la partie réseau prévient l'application de l'envoi.

Réception d'un paquet

La partie réseau lit en continu ses interfaces dans l'attente d'un paquet. Lorsqu'elle en reçoit un, on regarde si le paquet nous est bien destiné. Si oui alors on regarde si une application est “accrochée” au port de destination du paquet. Si oui, on lui envoie le datagramme. Sinon, c'est probablement une erreur d'envoi ou une tentative d'attaque.

En mode routeur, la partie réseau va faire suivre les paquets IPv6 sur ses interfaces, et, en IPv4 ou IPv6, va gérer la translation d'IP avec le DNS-ALG.

Construction du préfixe DNS-ALG

Le premier DNS-ALG du routeur est construit à partir de l'adresse Ipv6 interne du routeur.

Pour être conforme avec notre réseau, le préfixe DNS-ALG devra être de la forme : 2500:5247:1111::/96

V. Objets utilisés dans le simulateur

Classe Argument

Cette objet ne sert qu'au lancement de l'application pour parser le fichier de configuration et lancer la machine avec les bons arguments.

Classe Machine

Cette objet possède un nom d'hôte, une application réseau et un prompt.

Lors du lancement de la machine (méthode run), on ordonne a la partie réseau de se connecter. Ensuite, on va lancer un thread pour lancer le prompt. Lequel va partager une variable avec la machine pour propager les lignes tapées par l'utilisateur.

Classe CommandeMachine

Cette classe contient un ensemble de fonctions qui ne s'occupent pas du réseau. C'est une extension de la classe Machine.

C'est dans cette classe que sont stockées les fonctions *ifconfig* (qui lit la configuration réseau), *route* (qui affiche la table de routage), *verbose* (qui affiche ou change le niveau de verbosité selon l'argument donné) ou help (qui affiche toutes les commandes possibles).

Classe Creseau

Partie reseau de la machine.

Cette classe sert à redistribuer les ordres sur les interfaces concernées.

Classe Interface

Cette classe contient une interface réseau d'une machine, contenant une IPv4 et/ou une IPv6.

Classe Prompt

Gestion des interactions entre l'utilisateur et la machine.

Classes d'utilisation

Classe Ip

Cette objet ne contient que 2 attributs : une valeur et une version.

Cependant cette classe est très utile car elle permet de tester la validité d'une adresse IP avec la fonction *isValid* ou encore, dire si une IP est v4 ou v6 avec les fonctions repsectives *isIPv4* et *isIPv6*.

Classe Datagramme

Contient une IP source, une IP destination, un port source, un port destination, un champ TTL, un champ END et champ de données (DATA).

Un datagramme est envoyé sous forme de chaîne de caratères :

[76.9.1.1|45.22.145.2|89|53|2|0|blabla]

Classe LockFile

Cet objet permet de vérouiller un fichier en écriture par un système assez simple. Le but étant qu'il ne puisse pas y avoir deux processus écrivant en même temps dans le fichier.

Tout d'abord, le contructeur va prendre comme paramètre le nom du fichier.

Ensuite, la méthode *lock* va vérouiller le fichier et le méthode *unlock* va le déverouiller.

Méthode lock : On va regarder si le fichier composé du fichier à vérouiller et de l'extension “.lock” n'existe pas. Si il existe, on attend une seconde et on recommence. Si au bout de trois essais il existe encore, alors on renvoie 0.

Quand le fichier n'existe plus, alors on l'ouvre en écriture (le fichier est créé au passage), et on y place son **pid** à l'interieur, et on referme le fichier.

Ensuite, on va réouvrir le fichier en lecture, et vérifier que la première ligne qu'on trouve contient bien notre pid. Si oui, on revoie 1. Sinon, on peut réessayer 3 fois.

Ce mécanisme permet d'etre sûr qu'un fichier est vérouiller pour un seul processus.

Classe Routage

Permet simplement de gérer la table de routage du sous réseau concerné. Cet objet peut afficher la table de routage, ajouter et supprimer une ligne (en vérouillant le fichier).

Classe RouteLine

Reprensente un objet lu dans la table de routage. Il dispose de deux attrituts,

un nom d'hôte et une IP.

Classe Fifo

Objet permettant de gérer l'installation et la désinstallation du tube nommé dans le répertoire de l'interface concernée.

Application Reseau

Classe contenant une méthode run(datagramme).

Classe PingServer

Du fait que nous n'utilisons qu'un seul protocole, il a été choisi que le ping se ferait sur le port 0, il fallait donc créer un serveur de ping pouvant envoyer le pong .

Le serveur attend de recevoir un datagramme sur le port 0, il inverse le port source et destination, l'ip source et destination et il renvoie le datagramme sur le réseau.

Classe DnsServer

Le serveur DNS est ouvert sur le port 53.

Il attend de recevoir un datagramme. Dès qu'il en reçoit un, il lit le champ DATA qui doit être de la forme : *query AAAA hostname* ou *query A hostname*. Si cette forme n'est pas respectéz il renvoie un datagramme d'erreur à l'IP source. Sinon, il cherche une correspondance dans ses tables de routage.

Si il y a correspondance et qu'il s'agit d'IPv6, alors il renvoie l'adresse Ipv6. Par exemple : *query-response AAAA 2001:0000:0000::2*

Si il y a correspondance et qu'il s'agit d'IPv4, alors il crée une adresse DNS-ALG avec cette adresse v4 et re,voie l'adresse v6 demandée.

S'il n'y a pas correspondance, alors il renvoie : *query-response AAAA notfound* ou

query-response A notfound

Classe Ping

Il s'agit de la commande ping lancée par l'utilisateur.

Il va se connecter à un port libre sur la machine. On initialise la valeur NB à 1 s'il n'y a aucune valeur. Cette variable correspond au nombre de ping qu'on veut envoyer.

Si l'hôte est contactable directement, alors il va envoyer un paquet sur le port 0 de cette machine et attendre le résultat.

Sinon, il va envoyer une requète DNS sur le premier routeur trouvé et attendre la réponse.

Lorsqu'il reçoit un datagramme sur son port, après toutes les vérifications d'usage, on va regarder si on attend le résultat d'une requète DNS. Si oui, alors on regarde si le paquet nous renvoie bien un DNS.

Si le retour est *notfound*, alors l'application se termine (et libère le port).

Si la requète DNS a abouti ou si le paquet correspond a un pong, alors on décrémente NB. Si NB > 0, alors on envoie un ping sur l'hote. Et on attend de nouveau sur un paquet.

Sinon, l'application se termine et libère le port.

Classe Resolv

Cette application, tout comme Ping, va prendre en paramètre une machine hôte et un port libre sur la machine.

On va regarder si l'hôte est contactable directement, c'est-à-dire, si en lisant notre table de routage on trouve une correspondance. Si oui, on l'affiche et la commande se termine.

Sinon, on va envoyer une requète DNS sur le port 53 du premier routeur venu.

A réception d'un datagramme sur notre port, si le champ DATA correspond bien à un protocole DNS, alors on l'affiche et la commande se termine.

Sinon, on attend un autre paquet.

Conclusion

Le simulateur réalise tous les scénarios qui ont été demandés, mais il ne s'arrête pas là. La façon dont il a été conçu permet d'envisager d'autres scénarios. La fléxibilité des fichiers de configuration pourrait, par exemple, permettre d'insérer un second routeur très facilement.

Il est également modulable. D'autres fonctionnalités telles que nmap pourraient être implémentées en créant une nouvelle classe pour cette fonction.