Installation d'une passerelle Linux chez soi

Document version 1.0

création: 01/12/04

La lecture de ce document nécessite quelques connaissances du mon de IP et propose une installation simple et rapide d'un serveur Linux proposant une multitude de services. L'aspect réseau est détaillé dans certains chapitres notamment relatifs à la gestion du parefeu sous Linux. Le niveau nécessaire est « intermédiaire ».

Ce document évolue avec le site « http://www.ifrance.com/Raum/systeme/install_passerelle.html » et est disponible au téléchargement à l'adresse suivante :
« http://www.ifrance.com/Raum/systeme/install_passerelle.pdf »

me contacter: raum@forward.to

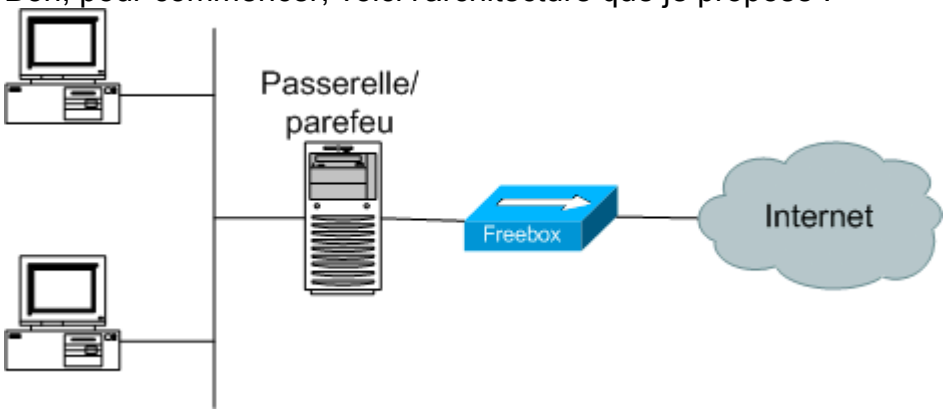
Table des matières

1.Architecture.....4
2.Installation de Linux (Mandrake 9.2b).....6
3.Partage de la connexion Internet.....8
3.1Fonctionnement.....8
3.2Comment ça marche ?.....8
3.3Autoriser l'IP Forwarding.....9
3.4Mettre en place la translation de l'adresse.....9
4.Configuration du service DHCP.....10
4.1Fonctionnement.....10
4.2Installation.....10
4.3Configuration.....10
4.4.....11
4.5Lancement du service.....11
5.Configuration du service Proxy.....12
5.1Fonctionnement.....12
5.2Installation d'Apache.....12
5.3Compilation et installation.....13
5.4Configuration.....13
5.5Démarrage du service.....14
6.Blacklister les sites de pub et autres.....15
6.1Fonctionnement.....15
6.2Remplissage du fichier "/etc/hosts".....15
7.Espace partagé sous Samba.....16
7.1Fonctionnement.....16
7.2Configuration de Samba.....16
7.3Démarrage du service Samba.....16
8.Configuration du service SSH.....18
8.1Fonctionnement.....18
8.2Démarrage du service.....18
8.3Et alors ? quoi d'autres ?.....18
9.Configuration du service OpenVPN.....20
9.1Fonctionnement.....20
9.2 Installation d'OpenVPN.....23
9.3Configuration d'OpenVPN.....24
9.4Génération du certificat authentifiant (ou root).....26
9.5Génération des certificats individuels.....27
9.6Configuration du serveur OpenVPN.....27
9.7Démarrage du serveur.....29
9.8Connexion depuis un client Windows.....31
9.9Lancement du serveur au démarrage.....32
9.10Et lancer le serveur avec xinetd ?.....35
10.Filtrages de plages d'adresses.....37
10.1Fonctionnement.....37

10.2Récupération du patch..... 37
10.3Installation d'iprange.....37
10.4Configuration et compilation du noyau..... 38
11. Parefeu sous Linux 39
11.1Fonctionnement..... 39
11.2Le filtrage Linux, comment ça marche ?..... 40
11.3Définition de la police par défaut.....42
11.4Exercice pratique..... 42
12. Configuration globale du parefeu.....46
12.1Connexion cliente depuis le parefeu.....46
12.2Connexion Serveur sur notre parefeu..... 47
12.3Connexion cliente d'une machine du LAN 48

1. Architecture

Bon, pour commencer, voici l'architecture que je propose :



La passerelle peut être une machine de type Pentium/AMD 300 à 500Mhz avec 128Mo. Les transferts entre l'ensemble des machines se fera à 100Mb/s en full duplex et vers la freebox a 10Mb. Voici ma machine perso :

- vendor_id : AuthenticAMD
- model name: AMD-K7(tm) Processor
- cpu MHz : 499.044
- cache size : 512 KB
- bogomips : 996.14
- Ddur : Western Digital 80Go (quasiment silencieux)
- Graphique : Xentor TNT2 16Mo AGP (là, problème car sans écran le pécé ne démarre pas !)
- Son : rien
- Distrib : Mandrake 9.2b2
- Alim : 250W
- Lecteur CD : Creative Labs x24 qui fait tac tac vroumf tzzzzz clac
- Lecteur disquette : pouhlà, un vieux vieux vieux
- Cartes ethernet : PCI NE2K, 10Mb HD/FD (possède même un connecteur BNC !!!) et une RealTek 8139, 10/100Mb HD/FD

Ma connexion à l'Internet passe par le réseau Free dégroupé. La connexion est donc permanente et le lien est de niveau 2. Donc pas de problème de PPP, de login chap, etc. On simplifie toute la mise en réseau de notre Linux.

La Freebox est connectée à ma "Linuxbox" par un câble droit RJ45 standard, sur la carte à 10Mb/s. La seconde interface ethernet, 100Mb/s Full Duplex, est connectée à un mini-commutateur, ainsi les transferts LAN auront 100Mb/s de bande passante.

Rappel:
HD signifie "Half Duplex", les machines discutent sur un seul canal et doivent attendre que l'une finisse de "parler" pour pouvoir prendre "la parole"... il peut y avoir des collisions, et il y a qu'un seul canal à 10 ou 100Mb/s. A lire toute littérature sur le protocole CSMA/CD.
FD signifie "Full Duplex", les machines ont deux canaux, un pour écouter, l'autre pour envoyer. C'est plus efficace car chaque canal dispose de 10 ou 100Mb/s.

2. Installation de Linux (Mandrake 9.2b)

Pour commencer, j'ai téléchargé cette distribution sur le site FTP de Jussieu.
La documentation Mandrake pour l'installation est disponible sur le site de Mandrake.

Lors de l'installation, je préfère utilisé l'outil de base "fdisk" en mode console mais la dernière version de DiskDruid est très bien... (pour passer en mode console lors de l'installation, appuyez sur Alt + F2 et lancer fdisk puis redémarrer l'install...)

Voici le système de partition que j'ai choisi, il est discutable (très...) mais j'ai décidé de pas faire compliqué... pas de "/var", pas de "/usr", pas de "/tmp" etc.

/dev/hda1	/	5Go
/dev/hda2	Swap	384Mo
/dev/hda3	Extended	
/dev/hda5	/home	384Mo
/dev/hda6	/space	70Go

Pour rappel, Linux utilise un système de fichiers type Unix. Il ne dipose pas des mêmes fonctionnalités qu'un système de fichiers type Windows. Plusieurs systèmes de fichiers sont disponibles, entre autres :

- Ext2FS, système de fichiers de base pour Linux
- Ext3FS, une extension de l'ext2FS apportant une journalisation des opérations sur le système de fichiers. Entre autre, cela permet de vérifier un système de fichiers très rapidement ! et cela limite la perte de données
- ReiserFS, nouveau système de fichiers (enfin nouveau...). Il est maintenant supporté par Linux directement au niveau du noyau. Il est sans doute plus rapide que l'ExtXFS.

Pour information, le NTFS de Windows 2K/XP sont journalisés mais pas plus robuste que l'Ext3FS ou le ReiserFS.

Pour l'installation de la distribution, j'ai coché :

- Machine de bureau
- Internet
- Ordinateur Réseau
- Configuration
- Développement
- Web/FTP
- Pare-feu/Routeur
- Serveur Réseau
- Station KDE

et choisir les paquetages individuellement. J'ai supprimé tout ce qui était son, j'ai rajouté certains serveurs, notamment :

- samba server
- dhcp server
- openssh server

Une fois arrivé à l'écran suivant :

Et j'ai configuré mes deux interfaces ethernet, en cliquant sur "Réseau Lan", (détection auto puis je supprime l'option bootp/dhcp), eth0 et eth1 étant différentes, je n'ai pas eu de mal à ne pas me tromper.

J'ai donné l'adresse 192.168.0.1 à mon interface LAN et mon IP fixe Free sur mon interface Freebox. A ce stade, ma machine démarre et surfe sur internet (je suis connecté par une freebox via une interface ethernet donc je simplifie et je n'expliquerais pas comment configurer tel ou tel autre modem particulier… certains sites proposent de très bons tutos…)

Rappel:

Les plages d'adresses suivantes sont dites privées :

- 10.0.0.0 - 10.255.255.255 (10/8)
- 172.16.0.0 - 172.31.255.255 (172.6/12)
- 192.168.0.0 - 192.168.255.255 (192.168 / 16)

Ces adresses ne sont pas routables, c'est à dire qu'elles ne sont pas présentes (donc pas joignables) sur l'Internet.

Il y a ensuite deux autres types d'adresses : les adresses publiques et les adresses multicasts.

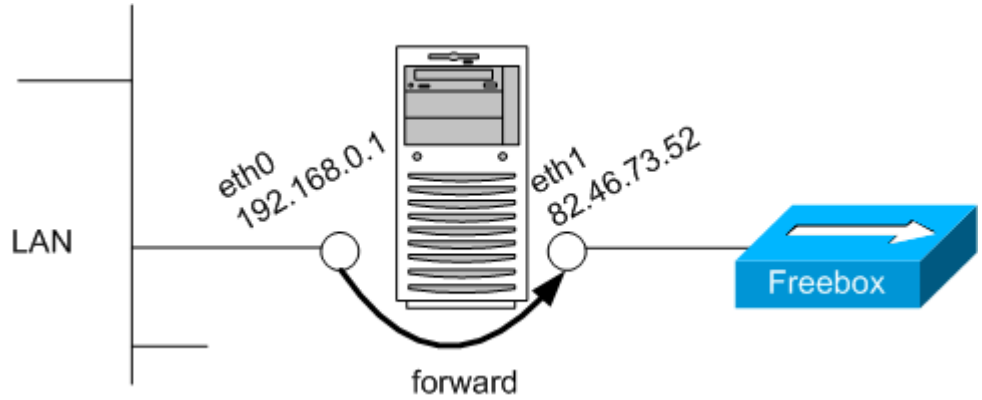
Lorsqu'on établit une connexion vers l'Internet, on obtient une adresse publique qui nous permet de naviguer sur la toile. Mon réseau local est un réseau privé, j'utilise donc des adresses privés conformément à la RFC 1918.

3. Partage de la connexion Internet

3.1 Fonctionnement

Alors, pour partager la connexion Internet comment faire ? Comment cela marche ?

Schématisons :

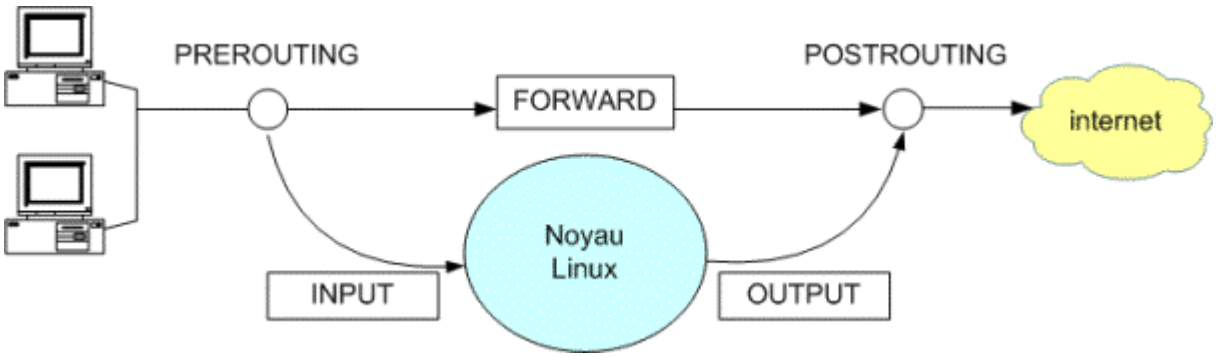


Une information qui est reçue doit "traverser" la passerelle. Depuis le LAN, l'interface "eth0" reçoit une trame et la transfère d'eth0 vers eth1 et inversement. C'est ce que l'on appelle "l'IP Forwarding". Ensuite, il faut changer l'adresse LAN, privée, par l'adresse internet, publique. C'est que l'on appelle la "translation d'adresse" ou encore "nattage" (qui vient de NAT qui signifie "Network Address Translation").

Donc pour partager la connexion, il faut :

1. autoriser les trames à traverser la passerelle
2. "natter" les adresses privées en adresses publiques

3.2 Comment ça marche ?



Une trame arrive par notre interface eth0 depuis notre LAN. Celle-ci arrive dans le noeud "PREROUTING". Soit nous souhaitons imposer un traitement à la trame, et dans ce cas-là nous l'enverrons dans l'INPUT du noyau, soit nous souhaitons que la trame traverse

passivement le pare-feu, et nous l'enverrons alors dans le FORWARD. En sortie du noyau, via l'OUTPUT, ou du FORWARD, la trame est envoyé au noeud POSTROUTING.

les "INPUT", "OUTPUT", et "FORWARD" sont appelés des "chaînes".On envoie les trames dans une chaîne et les ajoute à ces chaînes des règles pour interdire ou autoriser ces trames sur une chaîne. Mais pour le moment, on n'en est pas là...

LA commande utilisée pour gérer tout cela est la commande "iptables".

3.3 Autoriser l'IP Forwarding

Pour autoriser cette fonctionnalité, il suffit de taper la commande suivante dans une console en mode root :

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Explications: Linux propose un répertoire appelé "système de fichiers virtuels" et qui contient des fichiers virtuels, il s'agit du répertoire "/proc". La plupart des fichiers ont une longueur de zéro et ne nous intéressent pas. Par contre les répertoires "/proc/filesystems" et "/proc/sys/" contiennent des informations de configuration système. Ainsi, on trouve dans "/proc/sys/net/ipv4" des fichiers permettant de configurer la pile TCP/IP. Pour activer l'ip forwarding, il suffit que le fichier "ip_forward" contienne la valeur "1", et pour désactiver il suffit de mettre un zéro.

3.4 Mettre en place la translation de l'adresse

Ici, je suppose qu'il n'y a aucune règle de filtrage. Tout est autorisé sur les interfaces ethernet, en INPUT et en OUTPUT. Bref, la seule commande dont on a besoin pour la translation d'adresses est la suivante :

```
# iptables -A POSTROUTING -t nat -o eth1 -j SNAT --to 82.46.73.52
```

Et voilà.... normalement, un poste client qui aurait une adresse en 192.168.0.2 pourra sortir sur l'Internet.

Pour information, le poste client, Windows 2K/XP, doit être configuré avec :

- une adresse privé : 192.168.0.2
- l'adresse d'une passerelle, notre passerelle Linux en l'occurrence : 192.168.0.1
- les adresses des DNS, ceux de Free étant 212.27.32.176 et 212.27.32.177.

4. Configuration du service DHCP

4.1 Fonctionnement

En fait, j'ai configuré mes machines "fixes" avec une adresse fixe... 192.168.0.2 et 192.168.0.3. Ensuite lorsque que je connecte un ordinateur portable ou qu"un ami vient chez moi, il se connecte sur le commutateur et récupère une adresse dynamique, via le protocole DHCP (Dynamique Host Configuration Protocol, RFC 2131)

4.2 Installation

Il faut vérifier et installer, le cas échéant, les paquetages suivants : dhcp-server-3.0-1.rc12.3mdk,
dhcp-common-3.0-1.rc12.3mdk

```
# rpm -qa | grep dhcp
dhcp-server-3.0-1.rc12.3mdk
dhcp-common-3.0-1.rc12.3mdk
#
```

Et pour installer :

```
# cd /mnt/cdrom/Mandrake/RPMS
# rpm -i dhcp-server-3.0-1.rc12.3mdk dhcp-common-3.0-1.rc12.3mdk
```

Ils sont dispos sur les cédés de la mandrake (pour info, j'ai copié l'intégralité des RPM sur mon disque dur comme cela je n'ai plus besoin de chercher mes cédés et je peux aussi installer des paquetages à distance.... Ainsi, il m'a suffit de taper "rpm -i dhcp-common* dhcp-server* " et zou !)

4.3 Configuration

Ensuite, il suffit de modifier le fichier /etc/dhcpd.conf, voici le mien :

```
ddns-update-style none;

# ici, cela permet de récupérer les adresses de systèmes qui ne seraient
# plus connectés (lease time c'est le temps qu'attend le serveur DHCP pour
# contrôler qu'une adresse n'est plus utilisée, en gros)
default-lease-time 600;
max-lease-time 7200;

# ici, on définit les paramètres globaux de notre serveur DHCP

# en fait je pense que cette ligne est redondante mais je la laisse
option subnet-mask  255.255.255.0;

# on définit le domaine de broadcast (c'est la diffusion d'infos à l'ensemble
# du réseau)
option broadcast-address 192.168.0.255;
option domain-name  "chezmoi.fr";

# on définit les serveurs DNS qui seront envoyés à la machine qui récupère
# une adresse IP... Ainsi pas besoin de configurer les serveurs DNS à la
# main ! (donc ici, faut mettre vos propres serveurs DNS)
option domain-name-servers 212.31.34.172,212.31.34.173;

# ici, on définit la passerelle qui sera configurée sur les ordis,
# donc pas besoin non plus de définir la passerelle à la main !
option routers   192.168.0.1;

# et maintenant on définit enfin notre plage d'adresses....
# mon serveur DHCP peut attribuer 7 adresses
subnet 192.168.0.0 netmask 255.255.255.0 {
   range 192.168.0.4 192.168.0.10;
}
```

4.4 Lancement du service

Il suffit de lancer la commande disponible dans "/etc/init.d/dhcpd". Pour le lancer au démarrage, il suffit d'ajouter la commande dans le "rc.d".

```
# /etc/init.d/dhcpd start
# chkconfig --add dhcpd
```

La commande chkconfig a ajouté un lien dans le répertoire "/etc/rc.d/rc.3/S65dhcpd".

5. Configuration du service Proxy

5.1 Fonctionnement

J'ai préféré utilisé le service "proxy" proposé par le serveur Apache. En fait, les requêtes "HTTP" et "FTP" ne sont pas transmises depuis un poste client sur l'Internet mais l"envoi au serveur "proxy" qui va ensuite se charger de l'envoyer sur le net. Le serveur "proxy" peut appliquer des traitements à la requête.

5.2 Installation d'Apache

Je commence par supprimer les paquetages installés par la Mandrake, je télécharge le dernier "tarball" (tarball, c'est une archive "tar" ou une archive "tar" compréssée avec gzip ou bzip2) d'Apache puis je compile et installe.

Pour récupérer la liste des paquetages installés avec la mandrake :
```
# rpm -qa | grep apache
```

Pour supprimer les paquetages (dans mon cas) :
```
# rpm -e apache2-2.0.47-4mdk apache2-common-2.0.47-4mdk
```

Ensuite je télécharge et j'installe Apache :

Création d'un répertoire particulier où je stocke les fichiers sources
```
[/]# mkdir /compil
[/]# cd /compil
```

Récupération du tarball
```
[/compil]# wget http://apache.crihan.fr/dist/httpd/httpd-
2.0.48.tar.gz
```

Décompression du tarball
```
[/compil]# tar xzvf httpd-2.0.48.tar.gz

[/compil]# cd httpd-2.0.48
[/compil/httpd-2.0.48]#
```

5.3 Compilation et installation

On ajoute le module proxy au serveur apache

```
[/compil/httpd-2.0.48]#   ./configure   --enable-proxy   --enable-proxy-connect --enable-proxy-ftp --enable-proxy-http
```

On compile et on installe

```
[/compil/httpd-2.0.48]# make

[/compil/httpd-2.0.48]# make install
```

Apache est installé dans le répertoire "/usr/local/apache2". On aurait pu changer le répertoire de destination mais... boaaah... aller hop !

5.4 Configuration

Le fichier de configuration est "/usr/local/apache2/conf/httpd.conf".

Tout d'abord, j'ai modifié le port d'écoute qui est, par défaut, le port 80 par 8080.

```
# listen 80
listen 8080
```

Ensuite, je rajoute, à la fin du fichier, l'utilisation du module "proxy" :

```
<IfModule mod_proxy.c>
</IfModule>

ProxyRequests On
ProxyVia On

<Proxy *>
   Order deny,allow
   Deny from all
   Allow from 192.168.0
</Proxy>
```

On autorise seulement les postes du LAN privé à accéder au serveur Proxy. La ligne Order définit l'ordre de vérification des règles Deny et Allow. On utilise la règle Deny pour définir ce que l'on interdit, c'est à dire "tout" dans notre cas. Ensuite, on autorise les adresses privées. Ainsi une machine d'Internet, avec une adresse publique, ne pourra pas utiliser notre serveur ! Sans ces deux règles, n'importe qui pourrait utiliser notre serveur et "rebondir" sur notre passerelle pour surfer !

5.5 Démarrage du service

```
[/usr/local/apache2]# bin/httpd start

On vérifie que le serveur fonctionne
# netstat -apn | grep httpd
tcp    0      0 0.0.0.0:8080  0.0.0.0:*    LISTEN      1717/httpd
#
```

Pour lancer le service Apache au démarrage, il faut copier le script de démarrage dans le répertoire "/etc/init.d" et de créer des liens symboliques dans les répertoires "/etc/rc.d/rc3.d", "/etc/rc.d/rc4.d" et "/etc/rc.d/rc5.d". Ainsi en modifiant les niveaux d'exécution de Linux, le service serait démarré et arrêté correctement. Je suis fainéant, j'ai simplement rajouté une ligne dans le fichier "/etc/rc.d/rc.local" pour le démarrer....

Ensuite, il suffit de configurer l'explorateur Internet (FireFox, Netscape ou pire Internet Explorer) pour utiliser le serveur proxy...
Adresse du serveur proxy: 192.168.0.1
Port du proxy : 8080

Il est toujours possible de surfer sans proxy, les trames en destination d'un serveur Web sur le Net (adresse publique et port 80) partent du client, arrivent sur la passerelle et sont transférée, nattée et envoyée.

Si l'on veut supprimer cette possibilité, il suffit d'ajouter une règle dans un parefeu. Pourquoi ? Eh bien beaucoup de programmes qui tournent sous Windows tentent de se connecter sur des serveurs, par forcément des serveurs Web d'ailleurs, sur le port 80. On peut interdire cela...

```
# iptables -A FORWARD -p tcp -s 192.168.0.0/24 -d 80 -i eth0 -j DROP
```

	-A FORWARD: ajoute (append) à la chaîne FORWARD **-p tcp :** trame TCP (une trame TCP contient l'information des ports sources et destination) **-s 192.168.0.0/24 :** adresses IP source **-i eth0 :** interface d'entrée (input) **-j DROP :** envoi la trame à la fonction DROP (jump DROP), donc bêne la trame

Une trame qui arrive par l'interface de notre lan "eth0" et qui va en destination d'un port 80, quelque soit l'adresse de destination, est bênée. Maintenant, pour surfer, un client doit nécessairement utiliser le serveur proxy disponible à l'adresse 192.168.0.1 et sur le port 8080.

6. Blacklister les sites de pub et autres

6.1 Fonctionnement

Il existe un fichier système, sous Linux mais aussi sous Windows, qui permet de déclarer localement des noms d'hôtes.

Sur Internet, chaque machine possède une adresse IP, c'est à dire une adresse formée de quatre octets du style "82.46.73.52". Lorsqu'on tape une URL comme "www.google.fr", en fait on envoie pas de requête sur l'URL, il doit y avoir tout d'abord une résolution de nom pour trouver l'adresse IP associée à cette URL. Ensuite, la requête est envoyée à l'ordinateur associé à cette adresse IP.

Les FAI mettent à disposition deux serveurs de noms (ou serveurs DNS). Donc pour connaître l'adresse IP d'un serveur, on envoie la requête aux serveurs DNS. Pourtant, avant d'envoyer cette requête, le système regarde d'abord localement, dans un fichier de résolution de noms, si le nom d'hôte ne s'y trouve pas. Ce fichier est "/etc/hosts" sous Linux ou "c:\winnt\system32\drivers\hosts" sous Windows.

Le système va d'abord résoudre le nom avec ce fichier puis, s'il ne trouve rien, il envoie une requête de résolution aux DNS.

Si l'on met la ligne suivante dans le fichier "hosts" :

```
127.0.0.1 www.google.fr
```

Ensuite, lorsqu'on tape l'URL, la résolution du nom donne 127.0.0.1 et pas 66.102.11.104 ! Et l'adresse 127.0.0.1 est une adresse très spéciale car il s'agit de l'adresse locale de la machine, quand elle souhaite d'envoyer des messages à elle même elle utilise cette adresse IP. De fait, on ne pourra plus se connecter sur Google !

Eh bien pour blacklister les sites de pub, il suffit, au lieu de mettre "www.google.fr", de mettre des noms de serveurs de pub dans le fichier "hosts".

Lorsqu'un client souhaite se connecter sur une page HTML qui propose de la pub, la requête est envoyée au serveur proxy et le serveur proxy va faire la résolution de nom. La résolution va d'abord se faire par rapport au fichier "hosts" puis sur les serveurs de noms. Et c'est ainsi qu'on blacklistera les sites dont on ne veut pas !

6.2 Remplissage du fichier "/etc/hosts"

Vous pouvez touver un fichier "hosts" à cette adresse :
http://www.accs-net.com/hosts/how_to_use_hosts.html

Le mien contient plus de 19000 sites !

Voilà, c'est tout.... Ah ! une info, ne mettez surtout pas ce fichier sous Windows ! Vous risquez de bloquer votre machine :) (bah ouais... un Athlon XP1800+ a vu le processus "services" monté à 100% sous Windows alors que sur un 500Mhz sous Linux cela ne

pose strictement AUCUN problème, pas de charge... rien... :-/)

Espace partagé sous Samba

7. Espace partagé sous Samba

7.1 Fonctionnement

Le but ici est de disposer d'un espace disque partagé sur le LAN comme un répertoire partagé sous Windows. Le partage de ressources sous Windows utilise le protocole SMB et le programme SAMBA implémente le protocole SMB !

7.2 Configuration de Samba

Le fichier de configuration de samba est "/etc/samba/samba.conf".

```
On définit le groupe de travail déclaré sur toutes les machines du réseau
workgroup = DALINUX

On baisse le niveau de sécurité en partage, c'est plus facile
security = SHARE

On permet les invités mais seulement ceux ayant une adresse de notre LAN !
guest ok = yes
hosts allow = 192.168.0.
host deny = ALL

Et on partage le répertoire
[space]
comment = Espace passerelle
path = /space
browseable = yes
writable = yes
```

7.3 Démarrage du service Samba

Ensuite, il suffit de créer le répertoire "/space" et de relancer le service samba :

```
# mkdir /space
# /etc/init.d/smb restart

Pour ajouter Samba au démarrage du système
# chkconfig --add smb
```

Maintenant, le partage du dossier "/space" devrait apparaître dans le voisinage réseau des autres machines !

Nous ne permettons que les machines de notre réseau local à se connecter sur notre serveur Samba. Pourtant on peut ajouter une sécurité plus forte en paramétrant le parefeu !

Le protocole SMB utilise les port 137, 138 et 139 pour diffuser les différents partages.
Il faut donc rajouter les règles suivantes :

```
# iptables -A FORWARD -p tcp --sport 137:139 -o $OUTSIDE -j DROP
# iptables -A OUTPUT -p tcp --sport 137:139 -o $OUTSIDE -j DROP
# iptables -A FORWARD -p udp --sport 137:139 -o $OUTSIDE -j DROP
# iptables -A OUTPUT -p udp --sport 137:139 -o $OUTSIDE -j DROP
```

8. Configuration du service SSH

8.1 Fonctionnement

SSH signifie "Secure Shell". Pour simplifier, il s'agit en fait d'un terminal sécurisé. Cela permet de remplacer le "telnet" et cela permet de faire plein d'autres choses plutôt marrantes.

8.2 Démarrage du service

Le service est démarré par défaut. Pour contrôler, il suffit de taper les commandes suivantes :

```
# netstat -pan | grep sshd
tcp 0 0 0.0.0.0:22          0.0.0.0:*       LISTEN 1246/sshd
#
```

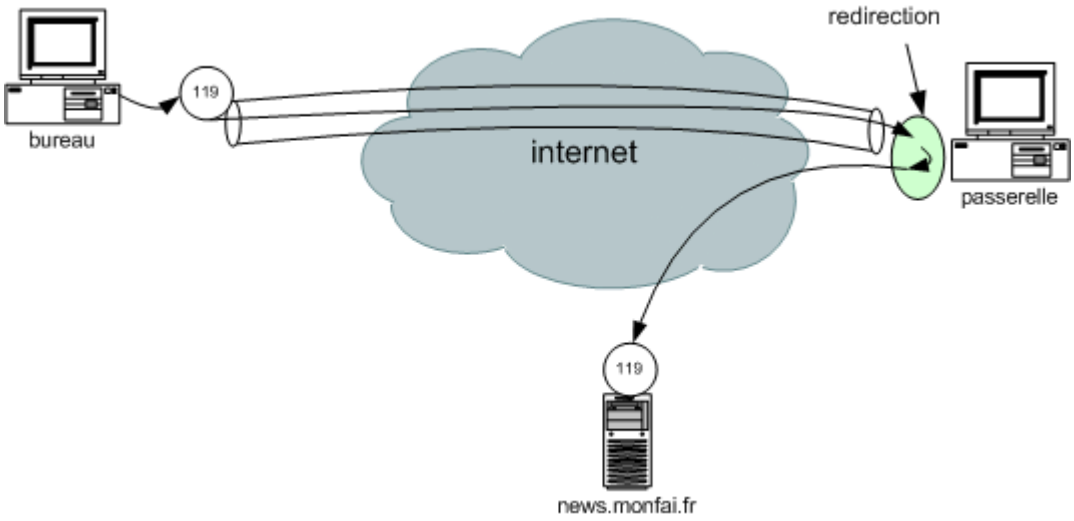
Ensuite pour se connecter à la station Linux depuis Windows, il suffit de télécharger l'excellentissime "PuTTY"... Site officiel du client Putty...

Là, il suffit d'exécuter "putty" et de sélectionner "SSH", taper l'adresse 192.168.0.1 et la connexion s'établit en mode sécurisé.

On se retrouve alors avec un shell sur notre passerelle !

8.3 Et alors ? quoi d'autres ?

Voici un schéma de principe :



Eh bien le principal intérêt de SSH, c'est de permettre la création de tunnels à l'intérieur de la connexion sécurisée ! C'est à dire ?

Prenons pas exemple la connexion au serveur de nouvelles. Je suis au bureau et je n'ai pas accès aux newsgroups. Par contre, j'ai le droit d'utiliser SSH et mon ordinateur à la maison peut se connecter sur le serveur de news de mon FAI....

Comment faire ? Eh bien, il suffit d'aller dans l'option "SSH" --> "tunnels" de putty et d'ajouter un tunnel lors de la création de la connexion sécurisée.

Voilà ce qu'il faut faire :

- taper "119" dans le boîte "source port" (il s'agit du port de connexion aux serveurs de news)
- taper "news.monfai.fr:119" pour rediriger les requêtes vers le serveur de news
- cliquer sur "Add"
- cliquer sur open
- s'authentifier normalement

Lors de l'établissement de la connexion SSH, un tunnel sera monté. Maintenant, il suffit d'indiquer au logiciel de lecture des news localhost comme serveur de news !

La boîte "source port" en fait va créer un port d'écoute sur la machine du bureau, en local et sur le port 119. Ensuite, on définit où se connecte l'autre bout de notre tunnel.

Notre newsreader va se connecter sur le port 119 de notre machine de bureau et le tunnel va envoyer ses requêtes à la passerelle qui va les rediriger vers le serveur de news du FAI !!!!

En fait, il est possible de tout faire avec cela ! Si l'on a pas de serveur SMPT, il suffit de créer un tunnel vers le serveur SMTP (port 25) de son FAI ! etc...

9. Configuration du service OpenVPN

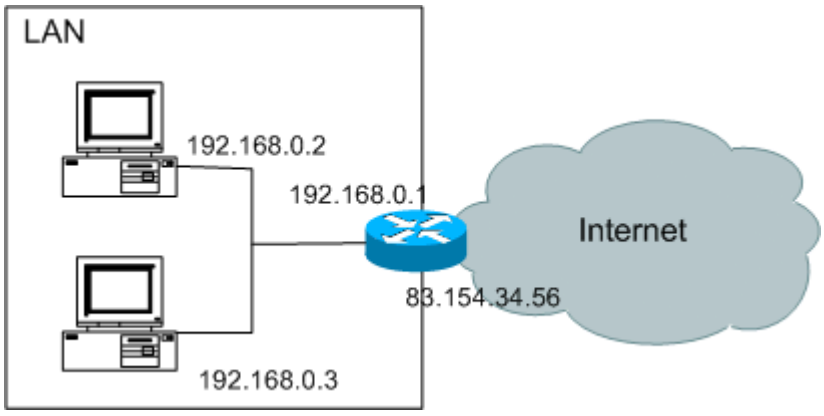
9.1 Fonctionnement

SSH signifie "Secure Shell". Pour simplifier, il s'agit en fait d'un terminal sécurisé. Cela permet de remplacer le "telnet" et cela permet de faire plein d'autres choses plutôt marrantes.

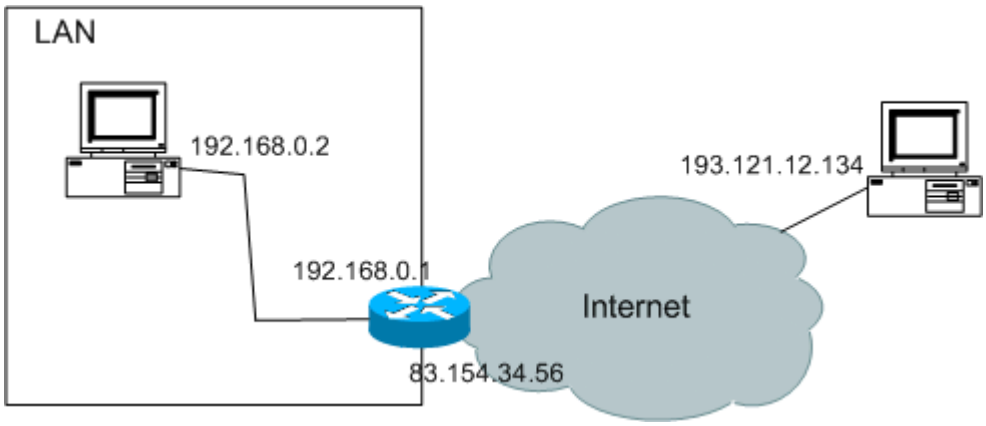
Bon, le VPN... comment cela marche ?

VPN est l'acronyme de Virtual Private Network et c'est le fait de "créer" un "réseau privé sur le réseau Internet public".

Par exemple, j'ai deux pécés à la maison qui sont reliés par un câble croisé, eh bien je peux leur mettre une adresse privée et réaliser des transferts de fichiers tranquille sous Windows. Il suffit de partager des répertoires puis de les déclarer d'une machine sur l'autre. Ces transferts se font en toute sécurité puisque l'on utilise le LAN. Le protocole utilisé est SMB, il s'agit d'un protocole utilisant les ports 137 à 139 et c'est du microsoft.



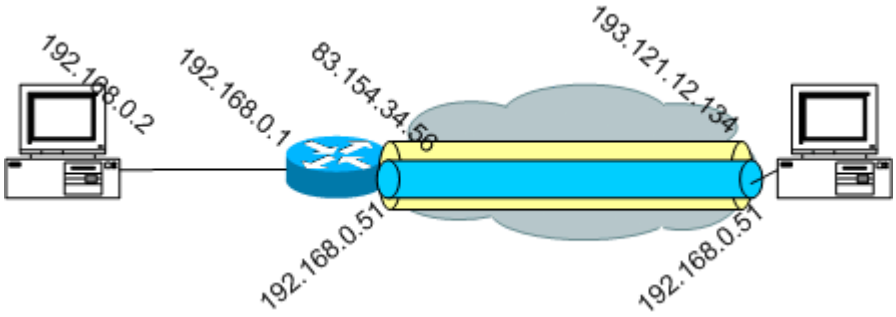
Bon, mettons que maintenant je suis sur un ordinateur portable, en déplacement, ou bien que je souhaite partager des fichiers avec mon frère qui se trouve à l'autre bout de la France, comment faire ?



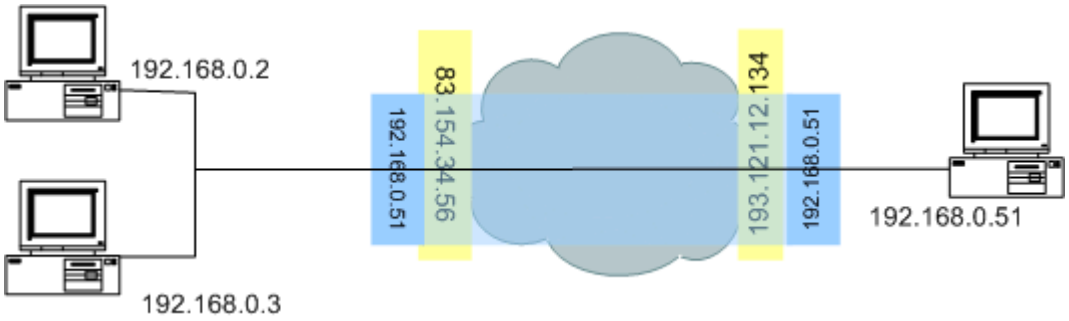
Dans ce cas-là, on peut utiliser tout autre type de protocole de transfert de fichiers comme par exemple FTP pour garder la possibilité de restreindre l'accès aux fichiers. Toutefois, il serait plus "pratique" de mettre à disposition les partages Windows (répertoires et imprimantes pourquoi pas ?). Comment faire ?

Eh bien le moyen est de raccrocher l'ordinateur distant au LAN pour qu'il croit qu'il se retrouve en LAN, même s'il traverse l'internet !
La solution pour réaliser cela s'appelle le VPN !

En gros, on créé un tunnel entre les deux extrémités de notre futur VPN et l'on assigne une adresse privée à chaque extrémité de ce tunnel.



Schématiquement, cela ressemble à cela :



L'ordinateur distant possède une interface physique qui a pour adresse 193.121.12.134 qui le connecte sur Internet. Ensuite, on construit un tunnel. Pour cela, on créé une interface virtuelle qui aura pour adresse 192.168.0.51. Sur la passerelle, on trouve le même schéma, c'est à dire une interface physique et une ou plusieurs interfaces virtuelles.

Ainsi, l'ordinateur distant se retrouve, virtuellement, sur le même LAN que les pécés à la maison ! si aucune règle n'est ajoutée sur le firewall pour empêcher la diffusion du protocole SMB, alors il est possible d'exporter les répertoires et imprimantes partagées ! A ce niveau de définition de l'architecture VPN, on peut utiliser tout protocole unicast de niveau 3, TCP ou UDP. Par définition, les broadcasts ne passent pas les routeurs ou, dans notre cas, la passerelle qui sert entre autres, de routeur.

Ce dernier point est embêtant dans le cas où nous souhaitons jouer en réseau. En effet,

la plupart des réseaux disposant d'une option de jeu en LAN diffuse des requêtes clients/serveur en broadcast ! Si notre passerelle, par défaut, ne permet pas de diffuser le broadcast, alors les jeux ne fonctionneront pas à travers le VPN.

Bon, mais ce qui nous intéresse, c'est de pouvoir jouer... alors comment faire ? eh bien la solution c'est de transformer notre routeur en pont ! (lire les articles à la page http://www.commentcamarche.net/lan/connect.php3 sur les ponts et routeurs, mais les explications sont un peu limitées...).

Bon, aller je réexplique différemment les ponts et routeurs. En fait, les deux équipements agissent à un niveau différent de la couche OSI. Pour rappel, la couche 1 est la couche physique, ce sont les câbles et interfaces électriques. Ensuite, nous trouvons la couche de liaison des données, couche 2, c'est ce protocole qui est utilisé dans les LAN, il s'agit en général du protocole ethernet. Ensuite, nous trouvons la couche 3 qui est une couche réseau, c'est la couche IP. En général, on associe le TCP et UDP à la couche 3.

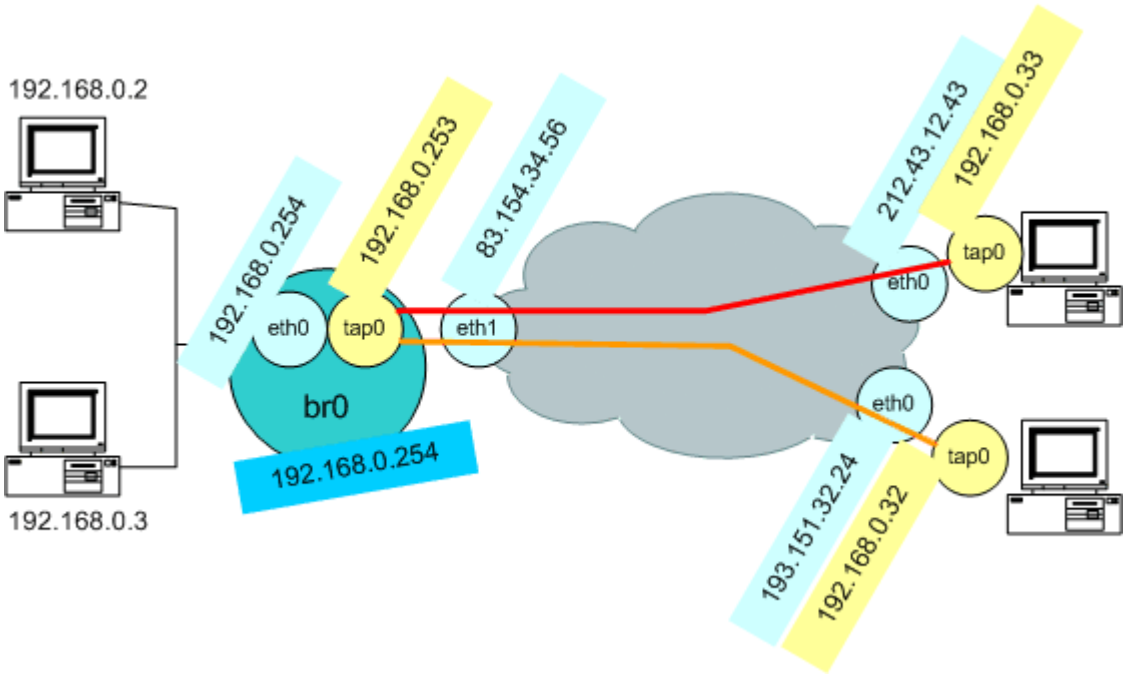
Donc en simplifiant, nous avons :

(NIV 1 : PHYSIQUE)
(NIV 2 : ETHERNET) (NIV 2 : TOKEN RING) .....
(NIV 3 : TCP / IP)

Un routeur agit au niveau 3, un pont au niveau 2. Un pont est, en général, utilisé pour transformer un protocole de niveau 2 en un autre protocole de niveau 2. Par exemple, certaines entreprises utilisent encore le protocole de niveau 2 propriétaire IBM "Token Ring". Pour pouvoir raccrocher ces LAN aux LAN ethernet, on utilise le pont pour diffuser les informations d'un LAN ethernet vers un LAN token ring et inversement.

Bon bref, revenons en à notre VPN. En fait, pour diffuser les broadcasts, il suffit de définir un pont virtuel sur la passerelle et d'y inclure l'ensemble de nos interfaces, virtuelles et physiques, puis de définir un domaine de broadcast commun.

On obtient quelque chose comme (bon, cela semble compliqué mais c'est pas si compliqué...)



L'interface virtuelle est nommée "TAP". On vient bien, par le schéma, les extrémités des tunnels et le fait qu'une principale interface virtuelle regroupe l'ensemble des interfaces.

9.2 Installation d'OpenVPN

Venons en maintenant à l'installation...

Tout d'abord, je propose d'utiliser une solution VPN très "light", il s'agit d'OpenVPN . Par rapport au schéma ci-dessus, je propose l'installation de la version 2.0 d'OpenVPN qui simplifie pas mal.... En fait, avec les versions inférieures, il fallait créer un serveur par client et ouvrir plusieurs ports d'écoute. Cette version permet la connexion de plusieurs clients sur le même serveur. Par contre la configuration est un peu plus compliquée... :-/

Donc les étapes sont simples : téléchargement des sources, décompression, compilation et installation.

A noter que pour utiliser une compression des données à la volée, vous pouvez aussi télécharger les librairies "LZO" disponibles sur le site www.oberhumer.com. Il faut télécharger, compiler et installer ces sources avant la configuration d'OpenVPN.

Vérification que la couche SSL est bien installée, si les librairies suivantes ne sont pas installées, eh bien... installez-les ! :)

```
# rpm -qa | grep ssl
openssl-0.9.7b-4mdk
libopenssl0.9.7-devel-0.9.7b-4mdk
libopenssl0.9.7-0.9.7b-4mdk
#
```

Récupération du tarball

```
[/compil]# wget
http://belnet.dl.sourceforge.net/sourceforge/openvpn/openvpn-
2.0_beta15.tar.gz
```

Décompression du tarball

```
[/compil]# tar xzvf openvpn-2.0_beta15.tar.gz

[/compil]# cd openvpn-2.0_beta15
[/compil/openvpn-2.0_beta15]# ./configure
[/compil/openvpn-2.0_beta15]# make
[/compil/openvpn-2.0_beta15]# make install
```

OpenVPN est installé dans le répertoire "/usr/local/sbin". Personnellement, j'ai créé un lien symbolique vers "/sbin", comme cela le programme est dans mon "path".

```
[/compil]# ln -s /usr/local/sbin/openvpn /sbin/openvpn
[/compil]# openvpn --version
OpenVPN 2.0_beta15 i686-pc-linux-gnu [SSL] [LZO] built on May 23
2004
Copyright (C) 2002-2004 James Yonan <jim@yonan.net>
[/compil]#
```

9.3 Configuration d'OpenVPN

Ca se complique car, eh bien, il faut installer tout un système de certificats alors qu'avec les versions précédentes, on pouvait utiliser une clé partagée. La gestion de cette couche est proposée par OpenSSL. Configurons donc OpenSSL !

Tout d'abord, retrouvé le fichier de configuration d'OpenSSL

```
[/root]# openssl ca
Using configuration from /usr/lib/ssl/openssl.cnf
<------ coupé ----->
[/root]#
```

Ok, donc éditons le fichier de configuration... J'ai quasiment tout laissé d'origine, excepté le répertoire où je vais stocker les certificats :

```
HOME = /usr/local/etc/ssl
...
[ CA_default ]

dir = /usr/local/etc/ssl # Where everything is kept
....
private_key = $dir/private/cakey.pem
....
```

Il faut aussi créer un périphérique TUN sous Linux :

```
[/root]# mkdir /dev/net
[/root]# mknod /dev/net/tun c 10 200
```

Rajouter la ligne suivantedans le fichier /etc/modules.conf :

```
alias eth1 ne2k-pci
alias eth0 8139too
probeall usb-interface usb-ohci
alias sound-slot-0 es1371

# Ligne à ajoutée pour la configuration du nouveau device au démarrage
alias char-major-10-200 tun
```

et enfin

```
On ajoute dynamiquement le module gérant les tunnels
[/root]# modprobe tun

Et on active le forwarding (voir la config du parefeu pour bien comprendre)
[/root]# echo 1 > /proc/sys/net/ipv4/ip_forward
```

9.4 Génération du certificat authentifiant (ou root)

On créé tout d'abord le répertoire où seront stockés les certifcats

```
[/root]# cd /usr/local/etc
[/usr/local/etc]# mkdir ssl
[/usr/local/etc]# cd ssl
[/usr/local/etc/ssl]# mkdir certs; mkdir private
[/usr/local/etc/ssl]# chmod 700 private
[/usr/local/etc/ssl]# echo '01' > serial
[/usr/local/etc/ssl]# touch index.txt
```

On génère maintenant le certifcat root avec un temps de validité très long

```
[/usr/local/etc/ssl]# openssl req -x509 -newkey rsa -out cacert.pem -outform PEM -days 10000
Generating a 1024 bit RSA private key
.++++++
.................++++++
writing new private key to 'privkey.pem'
Enter PEM pass phrase: <MOT DE PASSE>
Verifying - Enter PEM pass phrase: <MOT DE PASSE>
-----
...
-----
Country Name (2 letter code) [AU]:FR
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:Root Certif
Email Address []:
[/usr/local/etc/ssl]# ls
cacert.pem privkey.pem
[/usr/local/etc/ssl]# mv privkey.pem cakey.pem
```

Et l'on créé un fichier Dieffe-Helmann

```
[/usr/local/etc/ssl]# openssl dhparam -out dh1024.pem 1024
```

Maintenant, grâce à ce certifcat, notre machine pourra "signer" une requête de certificat d'une tierce personne.

Comment ça marche ? En fait, une personne va vouloir se connecter à notre VPN. Pour cela, il faut qu'elle s'authentifie avec un certificat. Ce certificat est envoyé à OpenVPN qui va contrôler sa validité, c'est à dire si ce certificat a été signé par notre certificat "root".

Pour avoir un certificat valide, il faut que l'utilisateur envoie une requête de certificat qui contient des informations personnelles ainsi qu'une clé personnelle. OpenSSL, par l'utilisation du certificat "root", va transformer cette requête de certificat en certificat, en ajoutant sa propre clé, des informations supplémentaires, etc. Ce certificat peut être renvoyé à la personne avec la clé publique de notre serveur certifiant (cacert.pem).

La clé "cacert.pem" doit être envoyée à toutes les personnes souhaitant s'authentifier auprès de notre serveur SSL.

9.5 Génération des certificats individuels

Ok donc.... nous avons un système qui permet la création de certificats... La marche à suivre est simple. Normalement, la requête doit être exécutée par la personne souhaitant le certificat... mais bon....

On génère un certifcat pour un ami, **l'option importante est "-node" pour permettre une authentification sans mot de passe**

```
[/usr/local/etc/ssl]# openssl req -newkey rsa:1024 -keyout testkey.pem -keyform PEM -out testreq.pem -outform PEM -nodes
Generating a 1024 bit RSA private key ..........................
++++++ .++++++
writing new private key to 'testkey.pem'
.....
Country Name (2 letter code) [GB]:FR
State or Province Name (full name) []:
Locality Name (eg, city) []:Chicago
Organization Name (eg, company) [My Company Ltd]:
...ETC...
Please enter the following 'extra' attributes to be sent with your certificate request A challenge password []:<MOT DE PASSE PERSO>
An optional company name []:
```

A ce stade, nous avons deux fichiers, la clé privé testkey.pem, et la requête de certificat. La clé est privée appartient à la personne requérant le certificat. On signe la requête de certificat....

```
[/usr/local/etc/ssl]# openssl ca -in testreq.pem -notext -out testcert.cert
<taper le mot de passe du certificat root !>
```

On peut supprimer la requête de certificat....

```
[/usr/local/etc/ssl]# rm testreq.pem
```

Maintenant, on a le certificat définitif "testcert.cert", il suffit d'envoyer la clé publique de notre serveur "cacert.pem", la clé privée de la personne "testkey.pem" et le certificat qui lie le tout "testcert.cert".

Bon, pour être tout à fait "secure" (comme on dit en ce moment), il faudrait que la clé privée soit générée directement par la personne, en bref je n'aurais pas du moi-même générer les clés des personnes qui viendront se connecter... Bref, aller pas grave !

9.6 Configuration du serveur OpenVPN

Pendant que nous y sommes, nous pouvons créer un certificat pour notre serveur VPN qui échangera son propre certificat. Au lieu de "test*.pem", je l'appelle "openvpn*.pem". On créé un répertoire "/etc/openvpn" et l'on y déplace les fichiers "openvpncert.cert",

"openvpnkey.pem" et l'on fait aussi une copie du fichier "/usr/local/etc/ssl/cacert.pem".

```
[/usr/local/etc/ssl]# mkdir /etc/openvpn
[/usr/local/etc/ssl]# mv openvpn* /etc/openvpn
[/usr/local/etc/ssl]# cp cacert.pem /etc/openvpn
[/usr/local/etc/ssl]# cd /etc/openvpn
```

Et maintenant créeons le fichier de configuration "server.conf" :

```
# FICHIER DE CONFIG : /etc/openvpn/server.conf
# version 1.1 : usage of new option "server-bridge" and inactivity control
#

# On définit le port d'écoute sur le port 5000
port 5000
# On définit le type d'interface virtuelle, on veut une interface ethernet virtuelle
dev tap0


ca /etc/openvpn/cacert.pem
cert /etc/openvpn/openvpncert.cert
key /etc/openvpn/openvpnkey.pem
dh /usr/local/etc/ssl/dh1024.pem

# je n'utilise pas d'encryptage, sinon je décommente la ligne suivante
#tls-cipher RC4-MD5


# la commande server-bridge remplace le bloc de commande suivant :
# mode server
# tls-server
# ifconfig-pool 192.168.0.32 192.168.0.64 255.255.255.0
# push "route-gateway 192.168.0.254"

server-bridge 192.168.0.254 255.255.255.0 192.168.0.32 192.168.0.64


# comme on veut créer un pont entre les différentes interfaces, on utilise

# des tunnels persistents. Franchement, je ne sais pas si c'est utile avec la

# version 2.0 mais avec la 1.5, il le fallait.... donc...
persist-tun
persist-key


inactive 3600
ping 10
ping-exit 60

user nobody
group nobody

verb 4
```

9.7 Démarrage du serveur

Maintenant, on créé notre interface virtuelle et puis on lance le serveur... J'ai créé un script pour tout cela... Il est simple et ne fait aucune vérification....

A noter, pour diffuser les broadcasts, lancés par certains jeux en mode réseau, il faut créer une interface virtuelle particulière que l'on appelle pont ou bridge.

Pour commencer, il faut installer les outils de "bridging", dans la mandrake 9.2b le paquetage a installé est le suivant :**bridge-utils-0.9.6-2mdk.i586.rpm**

```
[/root]# urpmi bridge-utils
```

Ensuite, il suffit de copier le script ci-dessous dans le répertoire "/etc/openvpn".

```
#! /bin/sh
#
# Created by Raum
# changes:
# 04/11/22 : daemon option used instead of daemon in config file
#

OPENVPN="/sbin/openvpn"

# On commence par creer un tunnel persistant en mode TAP
$OPENVPN --mktun --dev tap0

# on cree un nouvelle interface de type "bridge"
brctl addbr br0

# et on ajoute l'interface INTERNE du reseau local et l'interface
# virtuelle TAP dans le bridge
brctl addif br0 tap0
brctl addif br0 eth0

# on configure les interfaces en mode promiscuite (elles écoutent tout
# meme ce qui les concerne pas)
ifconfig eth0 0.0.0.0 promisc up
ifconfig tap0 0.0.0.0 promisc up

# enfin on configure notre interface virtuelle "bridge"
ifconfig br0 192.168.0.254 netmask 255.255.255.0 broadcast
192.168.0.255

# et on finit par demarrer OpenVPN.
$OPENVPN --daemon --config /etc/openvpn/server.conf --log-append /
var/log/openvpn.log
```

Maintenant, il suffit de lancer le script (n'oubliez pas de lui donner les droits d'exécution ;) chmod 700).
Ce script va créer une interface de type "pont" qui aura pour adresse 192.168.0.254

Pour information, voici le résultat de la commande "ifconfig" après lancement du serveur :

```
# ifconfig
br0 Lien encap:Ethernet HWaddr ##:##:##:##:##:##
   inet adr:192.168.0.254 Bcast:192.168.0.255 Masque:255.255.255.0
   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
...
eth0 Lien encap:Ethernet HWaddr ##:##:##:##:##:##
   UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
...
eth1 Lien encap:Ethernet HWaddr ##:##:##:##:##:##
   inet adr:83.154.34.56 Bcast:83.154.34.255 Masque:255.255.255.0
   UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
...
lo Lien encap:Boucle locale
   inet adr:127.0.0.1 Masque:255.0.0.0
   UP LOOPBACK RUNNING MTU:16436 Metric:1
...
tap0 Lien encap:Ethernet HWaddr ##:##:##:##:##:##
   inet adr:192.168.0.253 Bcast:192.168.0.255 Masque:255.255.255.0
   UP BROADCAST RUNNING PROMISC MULTICAST MTU:1500 Metric:1
...
```

9.8 Connexion depuis un client Windows

Commencez par télécharger le fichier installant OpenVPN 2.0 toujours sur le site d'OpenVPN.
Une fois OpenVPN installé, une nouvelle interface virtuelle a été créé dans "vos favoris réseau". Il suffit d'en ouvrir la propriété pour voir quelque chose comme "TAP-Win32 Driver".

Créez un répertoire "C:\VPN" et copiez les fichiers "testcert.cert", "cacert.pem" et "testkey.pem" puis créez un fichier de configuration :

```
# FICHIER DE CONFIG : C:\VPN\client.conf

# chez qui doit on se connecter et sur quel port ?
port 5000
dev tap
remote 82.46.73.52

# TLS parms
tls-client
ca c://VPN//cacert.pem
cert c://VPN//testcert.cert
key c://VPN//testkey.pem

# This parm is required for connecting
# to a multi-client server. It tells
# the client to accept options which
# the server pushes to us.
pull
ping 20
```

Puis, à partir d'une invite de commande, il suffit de taper :

```
C:\VPN> openvpn --config client.conf
```

Et dans une autre invite de commande :

On teste la joignabilité du pont

```
C:\> ping 192.168.0.254
Envoi d'une requête 'ping' sur 192.168.100.2 avec 32 octets de données :
Réponse de 192.168.0.254 : octets=32 temps<10 ms TTL=128
```

On teste la joignabilité d'une machine sur le LAN

```
C:\> ping 192.168.0.2
Envoi d'une requête 'ping' sur 192.168.100.2 avec 32 octets de données :
Réponse de 192.168.0.2 : octets=32 temps<10 ms TTL=128
```

9.9 Lancement du serveur au démarrage

Maintenant, la nouvelle adresse de la passerelle est 192.168.0.254, et plus 192.168.0.1

Vous pouvez aussi utiliser le script "openvpnd" suivant, à copier dans le répertoire "/etc/init.d".

```
#!/bin/bash
#
#
# chkconfig: 345 40 60
# description: Start and stop OpenVPN service
#
# Created by Raum v0.1 (last change 22/11/04)
# 22/11/04 : chkconfig compatibility
#

# Source function library.
. /etc/init.d/functions

test -x /sbin/openvpn || exit 0

RETVAL=0

prog="/sbin/openvpn"

start() {
    # Check if atd is already running
    if [ ! -f /var/lock/subsys/openvpn ]; then
        gprintf "Starting %s: " "$prog"
        daemon /etc/openvpn-2.0/start.sh
         RETVAL=$?
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/openvpn
        echo
    fi
    return $RETVAL
}

stop() {
    gprintf "Stopping %s: " "$prog"
    killproc /sbin/openvpn
    RETVAL=$?
    [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/openvpn
    echo
    return $RETVAL
}
```

```
restart() {
    stop
    start
}

reload() {
    restart
}

status_at() {
    status /sbin/openvpn
}

case "$1" in
start)
    start
    ;;
stop)
    stop
    ;;
reload|restart)
    restart
    ;;
condrestart)
    if [ -f /var/lock/subsys/openvpn ]; then
        restart
    fi
    ;;
status)
    status_at
    ;;
*)
    gprintf "Usage: %s {start|stop|restart|condrestart|status}\n" "$0"
    exit 1
esac

exit $?
exit $RETVAL
```

Le lancement est simple :

```
[/root]# chkconfig --add openvpnd
[/root]# /etc/init.d/openvpnd start
```

Et là, la vie est belle... On peut tout faire ! jouer à un jeu en réseau en utilisant l'option LAN, on peut partager des répertoires, on peut utiliser le proxy, etc.

Attention: pour la configuration de l'adresse du proxy, vous pouvez utiliser l'adresse 192.168.0.254

9.10 Et lancer le serveur avec xinetd ?

Le problème est simple. Avec la méthode si dessus, nous avons une instance d'OpenVPN toujours en fonctionnement. **Le nouveau mode "server" apporté par la version 2.0 ne permet pas le fonction avec xinetd.** Pour utiliser xinetd, il faut créer un tunnel par client et chaque tunnel est instancié par xinetd pour chaque client... bref, une usine à gaz.... Voici toutefois la marche à suivre... en gros. 1/ Ajout du service dans le fichier xinetd.conf

```
#
# Simple configuration file for xinetd
#
# Some defaults, and include /etc/xinetd.d/

defaults
{
instances = 60
log_type = SYSLOG authpriv
log_on_success = HOST PID
log_on_failure = HOST
cps = 25 30
}


service vpn
{
   type = UNLISTED
   port = 5000
   socket_type = dgram
   protocol = udp
   user = root
   wait = yes
   server = /sbin/openvpn
        server_args = --inetd wait --config /etc/openvpn/server.conf --log-append /
var/log/openvpn.log --inactive 600 --user nobody
}

includedir /etc/xinetd.d
```

Ensuite, pour voir en temps réel, les lancements des services :

une première fenêtre avec :

```
# tail -f /var/log/messages | grep xinet
```

une deuxième fenêtre avec:

```
# tail -f /var/log/messages | grep xinet
```

Si vous avez installé OpenVPN en tant que service, arrêtez le et supprimez le du démarrage automatique :

```
# /etc/init.d/openvpnd stop # chkconfig --del openvpnd
```

On relance le démon xinetd :

```
# /etc/init.d/xinetd restart
Arrêt de xinetd : [ OK ]
Lancement de xinetd : [ OK ]
#
```

On peut observer dans la première fenêtre de logs :

```
Nov 24 12:59:27 stargate xinetd[3408]: Exiting...
nov 24 12:59:27 stargate xinetd: Arrêt de xinetd succeeded
Nov 24 12:59:28 stargate xinetd[3432]: xinetd Version 2.3.11 started with
libwrap options compiled in.
Nov 24 12:59:28 stargate xinetd[3432]: Started working: 1 available services
Nov 24 12:59:30 stargate xinetd: xinetd startup succeeded
```

Et un petit contrôle supplémentaire pour vérifier que le port 5000 est bien ouvert en écoute pour le protocole UDP :

```
# netstat -pan | grep xinetd
udp 0 0 0.0.0.0:5000 0.0.0.0:* 3432/xinetd
```

Bon, normalement tout devrait être bon, reste plus qu'à lancer le client pour le test final.... et cela donne la chose suivante :

```
Wed Nov 24 13:50:20 2004 OpenVPN 2.0_beta15 i686-pc-linux-gnu [SSL] [LZO]
built on May 23 2004
Wed Nov 24 13:50:20 2004 Diffie-Hellman initialized with 1024 bit key
Wed Nov 24 13:50:20 2004 TUN/TAP device tap0 opened
Wed Nov 24 13:50:20 2004 /sbin/ifconfig tap0 192.168.100.253 netmask
255.255.255.0 mtu 1500 broadcast 192.168.100.255
....
Wed Nov 24 13:50:22 2004 81.254.0.230:5000 [Raum] Peer Connection Initiated
with 82.251.0.230:5000
Wed Nov 24 13:50:24 2004 Raum/82.251.0.230:5000 PUSH: Received control
message: 'PUSH_REQUEST'
Wed Nov 24 13:50:24 2004 Raum/82.251.0.230:5000 SENT CONTROL [Raum]:
'PUSH_REPLY,ifconfig 192.168.100.32 255.255.255.0' (status=1)
```

pour que cela fonctionne dans tous les cas, il faut modifier le fichier de configuration pour ne pas démarrer en mode "server" et créer un fichier de conf pour chaque utilisateur.... Bon courage...

10.Filtrages de plages d'adresses

Installation de la fonctionnalité IPRANGE pour iptables

10.1 Fonctionnement

Certains sites proposent des plages d'adresses appartenant à des origanismes que l'on a pas forcément envie de voir connecté sur notre passerelle, quoi que nous y fassions. Le but de ce chapitre est de permettre de récupérer ces fichiers et de les utiliser pour paramétrer note firewall.

Le premier problème, c'est que notre parefeu ne dispose pas d'une fonction pour traiter les plages d'adresses. On peut indiquer des sous-réseaux (c'est à dire adresse et masque) mais rien pour les plages !

Qu'à cela me tienne ! cette fonction existe mais elle n'est pas intégrée au noyau... Le but de la manoeuvre va être de récupérer le "patch", de recompiler le noyau et d'installer nos règles.

Le pré-requis sont, bien évidemment, les sources du noyau ! Vérifiez, par la commande "rpm -qa | grep kernel", que vous avez bien le paquetage suivant : "kernel-source-2.4.21-6mdk" (ou plus récent selon votre distribution...)

10.2 Récupération du patch

Les patches concernant le parefeu du noyau sont distribué par netfilter.org . On va télécharger le dernier "Patch O Magic" de Netfilter directement dans le répertoire "/usr/src" où doit se trouver les sources du noyau.

```
# cd /usr/src
[/usr/src/]# ls
linux@ linux-2.4.21-6mdk/ RPM/
[/usr/src/]# wget http://netfilter.org/files/patch-o-matic-20031219.tar.bz2
100%[====================================>] 293,822 295.81K/s ETA 00:00
[/usr/src/]# tar xvjf patch-o-matic-20031219.tar.bz2
[/usr/src/]# ls
linux@ patch-o-matic-20031219.tar RPM/
linux-2.4.21-6mdk/ patch-o-matic/
[/usr/src/]#
```

10.3 Installation d'iprange

Le patch-o-magic est un outil qui permet de "patcher" les sources du noyau pour corriger des bugs IP des filtres iptables et rajouter des fonctionnalités IP à la commande iptables. Le Patch O Magic contient moultes patches et est capable de déterminer ceux déjà intégrés dans les sources du noyau. Selon il propose d'en installer de nouveaux mais celui qui nous intéresse est le patch "iprange"....

```
[/usr/src/patch-o-matic]# cd patch-o-matic
[/usr/src/patch-o-matic]# ./runme base
```

Là, il peut dire **"Hey! KERNEL_DIR is not set"** et vous propose un répertoire cible "/usr/src/linux"... appuyez sur ENTREE... Les choses vont commencer à se gâter... En effet, patch-o-matic contient une multitude de patches et va proposer de les installer par paquet ou un par un... Le premier choix me propose d'appliquer 58 patches ! Si vous n'êtes pas sûr, répondez "no" à chaque question (en fait, appuyez sur ENTREE car la réponse par défaut est "no") jusqu'à tomber sur ça :

```
Testing... iprange.patch NOT APPLIED (2 missing files)
The base/iprange patch:
Author: Jozsef Kadlecsik <kadlec@blackhole.kfki.hu>
Status: Works

This patch makes possible to match source/destination IP
addresses against inclusive IP address ranges.

Examples.

iptables -A FORWARD -m iprange --src-range 192.168.1.5-192.168.1.124 -j ACCEPT
iptables -A FORWARD -m iprange --dst-range 10.0.0.0-10.5.255.255.255 -j ACCEPT
```

Là, répondez "yes" !!!

Ensuite, répondez "no" jusqu'à la fin.... Et le noyau a du être patché ! Reste plus qu'à le reconfigurer et le compiler....

10.4 Configuration et compilation du noyau

On sauvegarde l'ancienne configuration du noyau puis on lance l'outil de config...

```
# cd /usr/src/linux
[/usr/src/linux/]# cp .config config-DATE.bak

SI VOUS ETES EN TEXTE

[/usr/src/linux/]# make menuconfig

SI VOUS ETES EN GRAPHIQUE

[/usr/src/linux/]# make xconfig
```

11. Parefeu sous Linux

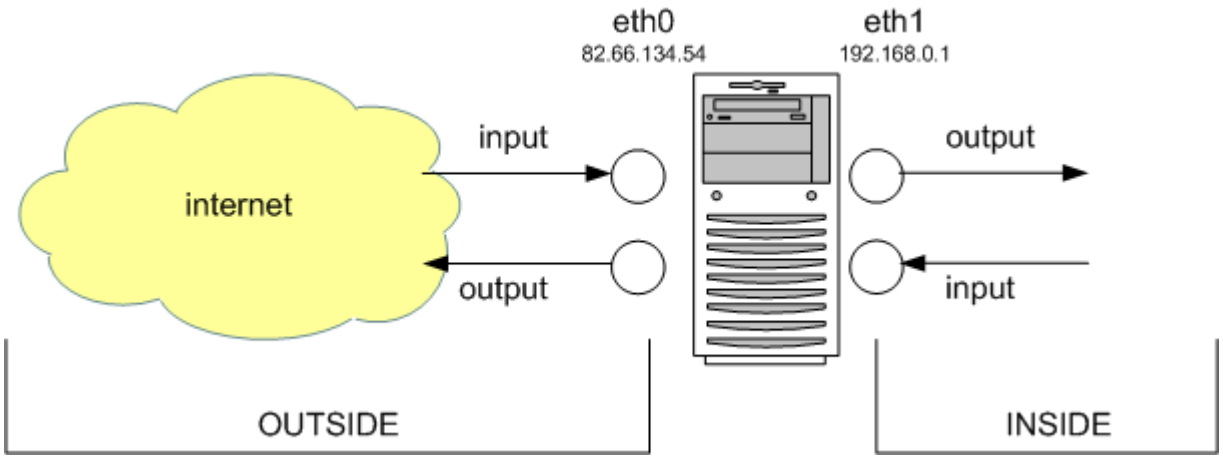
11.1 Fonctionnement

Dans ce chapitre, on va approfondir le fonctionnement du parefeu Linux et construire un vrai bastion. Le but du parefeu est de réceptionner, filtrer et envoyer des trames IP. On pourra filtrer les protocoles TCP/IP, UDP/IP et ICMP. Avec ce parefeu, nous pourrons protéger notre LAN de vilains qui rôdent sur l'Internet...

Dans un chapitre précédent, nous avions configuré le parefeu avec des règles relativement permissives. Maintenant, nous allons cloisonné le parefeu et contrôler l'ensemble des flux entrants et sortants.

En bref, on définit une règle par défaut : rien ne rentre, rien ne sort. On autorisera au cas par cas... C'est très contraignant mais c'est ce qui pourra assurer que notre parefeu est totalement sûr.

Mais un parefeu sous linux, comment cela fonctionne ?



Tout d'abord, nous allons définir deux zones :

- la zone "OUTSIDE" qui est la zone non sûre (là où rôdent les vilains pirates)
- la zone "INSIDE" qui est la zone sûre, notre LAN

Parfois on entend parler de "DMZ" ce qui signifie "DeMilitarized Zone", c'est une zone tampon qui est accessible depuis l'Internet et depuis le LAN. On y place des serveurs qui seront potentiellement attaquées (même régulièrement). Il s'agit d'un second LAN moins sécurisé que l'INSIDE. Dans notre architecture, on n'utilise pas de DMZ.

11.2 Le filtrage Linux, comment ça marche ?

Le noyau Linux possède des fonctionnalités de filtrage de trames. Une trame arrive, que ce soit par l'une ou l'autre des interfaces ethernet, le parefeu va regarder si l'on a écrit une règle particulière pour cette trame et va appliquer la règle s'il y en a une.

Pour rappel, un paquet IP contient les informations nécessaires au routage du paquet, c'est à dire **adresse IP source** et **adresse IP destination**. Sur cette couche IP, on construit une couche session qui peut être TCP ou UDP. Cette couche apporte d'autres informations comme le **port source** et le **port destination**.

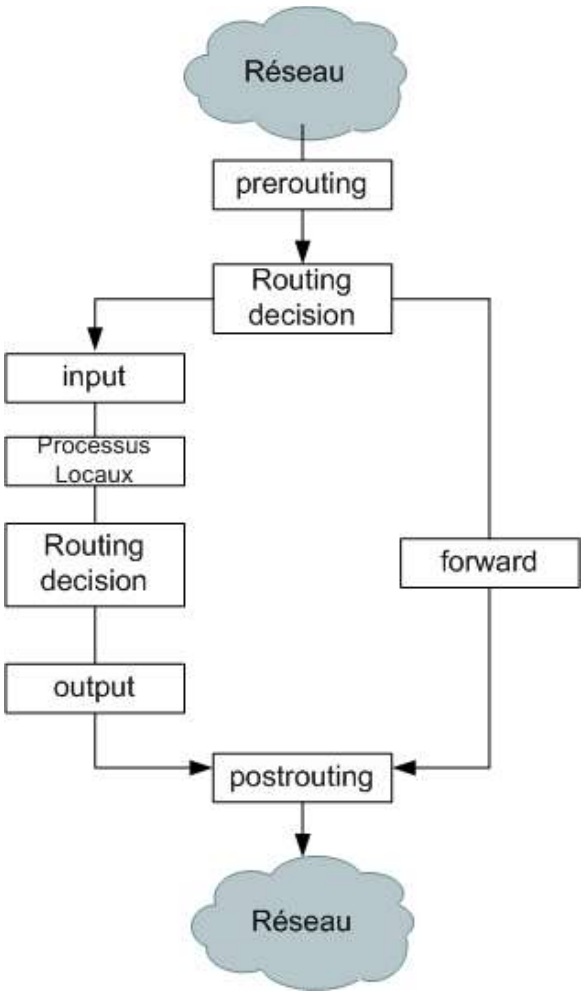
Port source, port destination, mais c'est quoi tout ça, je comprends pas ?

Ehm... Bon prenons par exemple un immeuble. Dans cette immeuble sans boîte à lettre, tu as un cordonnier, un forgeron et un informaticien (si on a toujours besoin d'un informaticien pour changer une ampoule). Tu as un trou dans tes chaussures et tu as donc besoin des services du cordonnier. Déjà, il va falloir que tu partes de chez toi pour aller jusqu'à cet immeuble, tu utilises l'adresse de l'immeuble pour trouver ta route. Une fois devant cet immeuble, si tu ne sais pas à quel étage se trouve le cordonier, tu vas rester planté devant l'immeuble... Par contre, si tu sais qu'il est au deuxième étage, tu vas aller taper à la bonne porte et le cordonnier pourra arranger tes chaussures. Ensuite tu repartiras chez toi en utilisant ton adresse à toi pour trouver ta route. Et puis tu remonteras à ton étage pour rentrer che toi. Les autres personnes de ton immeuble pourront aussi chercher les services de l'immeuble des services.

Ok, belle analogie mais ça donne quoi sur le net ?

Eh bien, tout d'abord mettons que nous avons sur le net un serveur qui propose un service FTP et un service HTTP (ou web). Ta machine et le serveur ont une adresse IP ce qui fait que tu pourras être "routé" de chez toi jusqu'à ce serveur. Une fois arrivé au serveur, grâce à l'adresse IP, tu vas demander un service spécifique, le service 21 pour le FTP ou le service 80 pour le HTTP. En fait, on appelle cela des ports et ils sont normalisés... Donc le port 20 et 21 sont les ports utilisés par le protocole FTP et le port 80 est utilisé par le protocole HTTP. Un programme spécifique "écoute" un port spécifique. Un serveur Web va écouter sur le port 80 et un serveur FTP sur le port 21. Un paquet IP arrive, le serveur va regarder l'entête TCP pour savoir vers quel port envoyer le paquet et donc vers quel programme. Le programme va prendre en compte la requête et y répondre par un paquet qui aura pour adresse source l'adresse du serveur, port source le port demandé par l'utilisateur et pour adresse destination ta machine et port destination le port ouvert par le programme qui a envoyé la requête.

Revenons maintenant au filtrage à proprement parlé... Linux utilise ce que l'on appelle des "chaînes" pour stocker les règles de filtrage. Un paquet est envoyé dans une chaîne où il est filtré en fonction des règles qui y ont été insérées.



En fait, que ce soit depuis l'Internet vers le LAN ou l'inverse, ou encore depuis l'Internet vers l'Internet ou depuis le LAN vers le LAN (par exemple si l'on fait tourner un serveur web sur notre parefeu, le paquet vient de l'Internet, est traité par notre serveur web et renvoyé sur l'Internet). Un paquet arrive sur notre parefeu. Il arrive tout d'abord dans le noeud PREROUTING où l'on peut exécuter certaines tâches (comme la translation d'adresses ou nattage).

Ensuite si le paquet est à destination d'une adresse de notre parefeu (82.46.73.52 depuis l'OUTSIDE ou 192.168.0.1depuis l'INSIDE) alors le paquet est envoyé dans la chaîne "INPUT". Sinon le paquet est envoyé dans la chaîne "FORWARD".

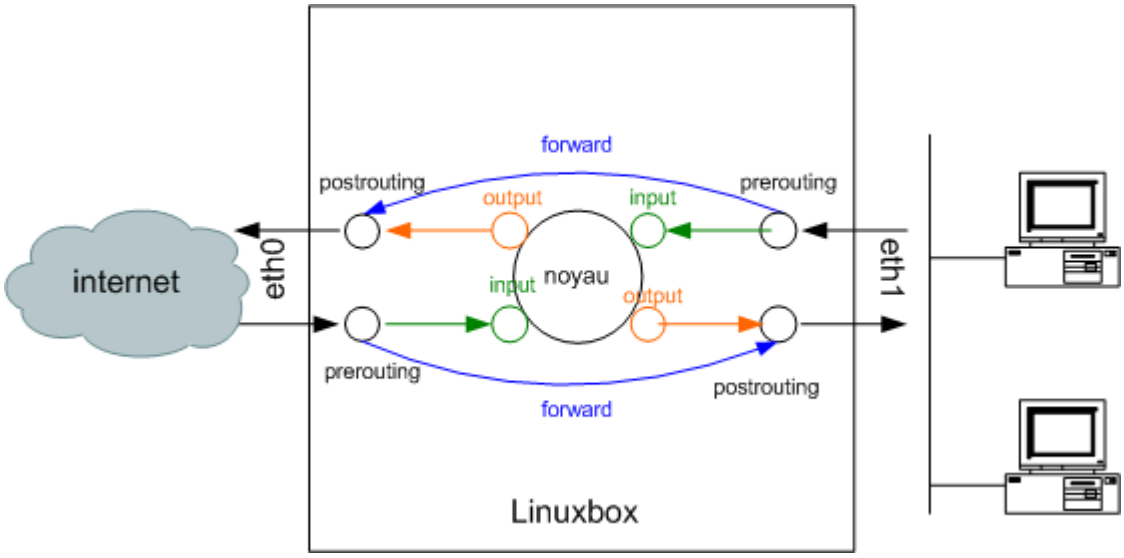
Si aucune règle n'a rejeté le paquet dans la chaîne INPUT, alors le paquet est traité par les processus locaux (par exemple un serveur WEB !) et envoyé dans la chaîne "OUTPUT" où il est filtré de nouveau.

Enfin les paquets des chaînes FORWARD et OUTPUT sont envoyés dans le noeud "POSTROUTING" où l'on peut exécuter certaines tâches (comme la translation d'adresses).

Je répète, un paquet est envoyé sur la chaîne FORWARD quand les adresses IP ne correspondent pas à l'une des adresses de notre serveur. Par exemple, depuis mon LAN je cherche à joindre le serveur de google. Je vais envoyer un paquet avec comme adresse IP source 192.168.0.2 et comme adresse destination 66.102.9.104. Le paquet arrive en PREROUTING puis il est routé en fonction de l'adresse destination. Linux regarde l'ensemble des adresses IP qu'il gère, 192.168.0.1 et 82.46.73.52, et décide que 66.102.9.104 n'appartient pas à ses adresses, il l'envoie dans la chaîne FORWARD et ce paquet n'arrivera jamais jusqu'aux programmes qui tournent sur notre parefeu. Ce paquet ne doit pas être traité par notre serveur mais par le serveur de Google.

Si l'on fait tourner un serveur Web sur notre parefeu, un paquet arrive avec pour adresse destination 82.46.73.52, Linux détermine qu'il s'agit d'une adresse qu'il gère et donc il l'envoie dans la chaîne "INPUT" et pas "FORWARD". Ensuite ce paquet, s'il n'est pas rejeté, arrive jusqu'aux processus où un prendra et traitera la requête pour renvoyer la réponse dans la chaîne "OUTPUT" puis renvoyé sur le net.

Schématiquement, on a (à peu près) ça :



11.3 Définition de la police par défaut

Commençons... Nous souhaitons que rien ne sorte et ne rentre sans notre permission. On va définir une "police" par défaut comme suit :

```
# iptables -P INPUT DROP
# iptables -P OUTPUT DROP
# iptables -P FORWARD DROP
```

	-P INPUT: police par défaut sur la chaîne INPUT **DROP:** on bêne

On dispose de plusieurs "fin" pour les trames :

- DROP : on bêne sans concession, sans état d'âme
- REJECT : on rejette le paquet et l'on avertit la machine qui nous a envoyé le paquet.
- ACCEPT : on accepte les paquets

11.4 Exercice pratique

Dans l'état, la connexion Internet est partagée, nous avons activé le "FORWARDING" mais dans l'état actuel des choses, toute trame qui arrive dans la chaîne INPUT ou FORWARD est bênée.

Mettons que nous souhaitions permettre aux machines de notre LAN d'aller se connecter sur le serveur de google (66.102.9.104). Dans un premier temps, nous ne nous occupperons pas de la résolution DNS et l'on tapera comme adresse "http://66.102.9.104" au lieu de "http://www.google.fr".

Voilà ce que je vois sur mon parefeu avec l'utilitaire TCPDUMP (qui permet d'afficher l'ensemble des paquets, je renomme ETH0 en INSIDE et ETH1 en OUTSIDE) et je

simplifie largement :

```
INSIDE :
192.168.0.2.1105 > 66.102.9.104.http
192.168.0.2.1105 > 66.102.9.104.http
OUTSIDE:
```

On voit que les paquets arrivent sur notre interface INSIDE mais ne sont pas transférées sur l'interface OUTSIDE. En effet, par défaut on bêne, il faut donc autoriser les paquets à destination d'un serveur Web à passer de l'INSIDE à l'OUTSIDE:

On autorise les paquets à être envoyée dans la chaîne FORWARD. On doit l'activer, une seule fois suffit

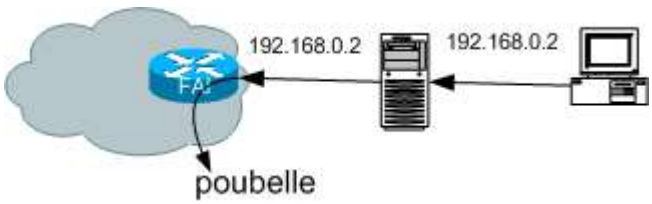
```
# echo 1 >/proc/sys/net/ipv4/ip_forward
# iptables -A FORWARD -p TCP --dport 80 -j ACCEPT
```

-A FORWARD: on ajoute une règle à la châine FORWARD
-p TCP : la règle concerne le protocole TCP, nécessaire pour pouvoir utiliser l'argument --dport 80
--dport 80 : destination port 80, on spécifit le port destination du paquet
-j ACCEPT : on accepte de laisser le paquet

Lors du filtrage, un paquet qui souhaite aller vers un serveur WEB (utilisation du protocole TCP et le port du serveur Web est 80), ce paquet est accepté.

Voici ce que l'on voit sur le parefeu maintenant :

```
INSIDE :
192.168.0.2.1105 > 66.102.9.104.http
192.168.0.2.1105 > 66.102.9.104.http
OUTSIDE:
192.168.0.2.1105 > 66.102.9.104.http
192.168.0.2.1105 > 66.102.9.104.http
```



Donc notre parefeu envoie sur le net une adresse privée ! non routable ! Nous n'obtiendrons jamais de réponse car les routeurs des FAI sont configurés pour rejeter tout paquet ayant pour adresse source une plage d'adresses privées. Eh oui, chez moi j'utilise 192.168.0.1, mais tout le monde peut utiliser ces adresses privées ! Ces adresses ne sont pas uniques et c'est pourquoi elles ne sont pas routées.

Pour remédier à cela, on met en place une translation d'adresse. C'est à dire qu'avant de sortir de notre interface OUTSIDE, on va remplacer l'adresse 192.168.0.2 par l'adresse Internet de notre parfeu. Automatiquement, le parefeu va noter cette translation d'adresse et lorsque le paquet revient, il changera son adresse publique par l'adresse privée et l'enverra sur le LAN.

```
# iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth1 -j SNAT --to 82.46.73.52
```

	-t NAT: type de service, NAT (Network Translation Adress) **-A POSTROUTING:** on ajoute le service de NAT au noeud POSTROUTING **-s 192.168.0.0/24 :** pour toutes les adresses sources appartenant au réseau 192.168.0.0/24 (c'est à dire 192168.0.0 -> 192.168.0.255). **-j SNAT :** on saute (jump) au service de NAT static **--to 82.46.73.52 :** et l'on transforme l'adresse du LAN avec l'adresse 82.46.73.52

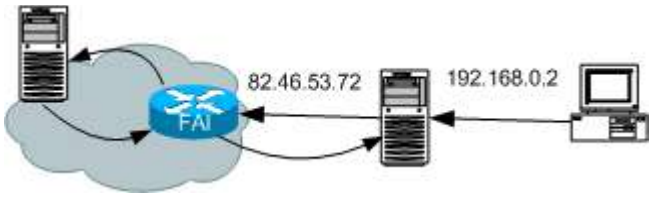
deuxième méthode pour le nattage :

```
# iptables -A POSTROUTING -t nat -o eth1 -j MASQUERADE
```

Cette forme est plus simple que la précédente, le résultat est le même... Simplement, on laisse le noyau Linux décider de la politique de translation d'adresse tout seul comme un grand.

Et voilà ce que l'on obtient :

```
INSIDE :
192.168.0.2.1105 > 66.102.9.104.http
192.168.0.2.1105 > 66.102.9.104.http
OUTSIDE:
82.46.73.52.3544 > 66.102.9.104.http
66.102.9.104.http > 82.46.73.52.3544
```



Ah ! Grande évolution ! On voit bien la translation d'adresse ! et le serveur répond ! Mais le paquet n'est pas retransmit de l'OUTSIDE vers l'INSIDE, de fait notre explorateur continue de lancer sa requête.... mais ne reçoit jamais de réponse...

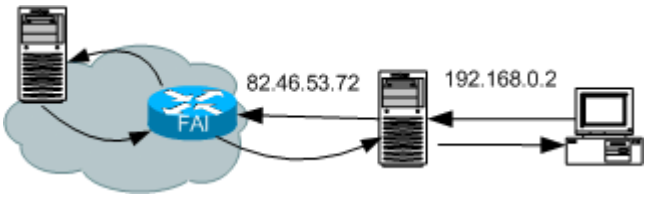
Pour cela, que faut il faire ? Eh bien comme on a permit à la trame de passer de l'INSIDE vers l'OUTSIDE, il faut maintenant s'occupper de la traversée inverse. A noter que les ports source et destination ainsi que adresses IP source et destination ont été interverties !

```
# iptables -A FORWARD -p TCP --sport 80 -j ACCEPT
```

	-A FORWARD: on ajoute une règle à la châine FORWARD **-p TCP :** la règle concerne le protocole TCP, nécessaire pour pouvoir utiliser l'argument --sport 80 **--sport 80 :** source port 80, on spécifit le port source du paquet, qui est 80 (voir dernières traces) **-j ACCEPT :** on accepte de laisser le paquet

On accepte de laisser passer les trames en provenance d'un serveur web dans la chaîne FORWARD. Et voilà le résultat !

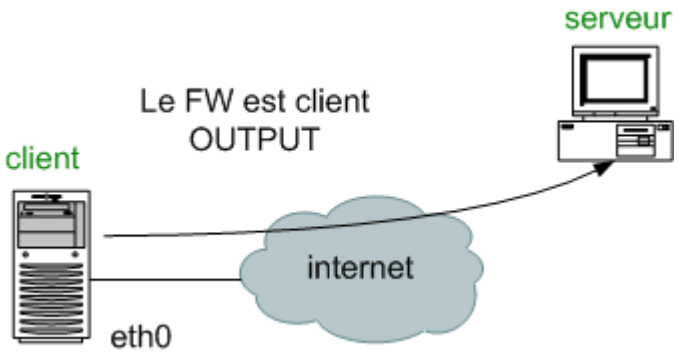
INSIDE :
192.168.0.2.1105 > 66.102.9.104.http
66.102.9.104.http > 192.168.0.2.1105
OUTSIDE:
82.46.73.52.3544 > 66.102.9.104.http
66.102.9.104.http > 82.46.73.52.3544



On voit que la translation d'adresse inverse est automatique. Le paquet traverse dans un sens et dans l'autre... Tout fonctionne ! Mais seulement pour le web....

12. Configuration globale du parefeu

12.1 Connexion cliente depuis le parefeu



Nous souhaitons, par exemple, pouvoir nous connecter sur un serveur Web comme "www.google.fr". Que se passe t'il ? nous envoyons une trame TCP/IP vers l'adresse "www.google.fr"sur le port 80, qui est le port normalisé HTTP. Nous sortons... Notre police, par défaut, c'est de refuser tout ce qui sort, donc depuis notre linux, nous ne pourrons pas joindre le serveur Web de google.... Il faut ajouter une règle pour dire "tout ce qui sort pour se connecter sur un serveur Web à le droit de sortir".... "Tout ce qui sort", on va donc rajouter une règle dans la chaîne OUTPUT... "pour se connecter sur un serveur Web", donc quand on va vouloir se connecter sur le port 80 par le protocole TCP (car le protocole HTTP est basé sur TCP, pas UDP) "doit être accepté", on spécifie que ce type de trame doit être acceptée. Ce qui donne, en langage iptables :

```
# iptables -A OUTPUT -p TCP --dport 80 -j ACCEPT
```

	-A OUTPUT: on ajoute une règle à la châine OUTPUT **-p TCP :** la règle concerne le protocole TCP, nécessaire pour pouvoir utiliser l'argument --dport 80 **--dport 80 :** destination port 80, on spécifit le port destination du paquet, qui est 80 **-j ACCEPT :** on accepte de laisser le paquet

Problème ! Le serveur Web va répondre ! et donc une trame va être retournée à notre parefeu et comme la police par défaut pour tout ce qui arrive est de bêner, notre trame sera bênée et on ne pourra toujours pas se connecter sur le serveur Web ! Eh oui, il faut se TOUJOURS se soucier de l'aller comme du retour ! Enfin, on aurait pu tout aussi bien changer la politique par défaut en acceptant tout ce qui rentre ou tout ce qui sort mais dans ce cas-là, il n'y a plus d'intérêt....

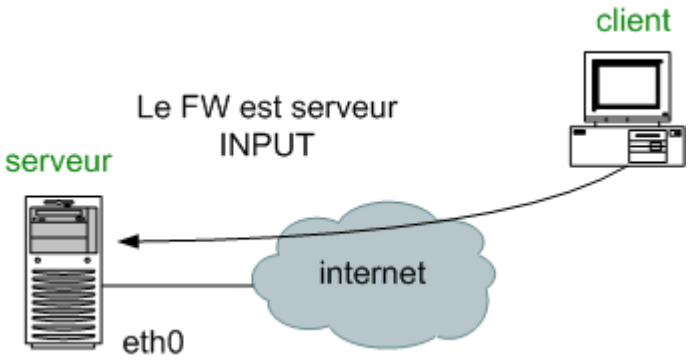
Donc on veut accepter les trames qui ont pour origine un serveur Web... ce qui donne en langage iptables :

```
# iptables -A INPUT -p TCP --sport 80 -j ACCEPT
```

-A INPUT: on ajoute une règle à la châine OUTPUT
-p TCP : la règle concerne le protocole TCP, nécessaire pour pouvoir utiliser l'argument --sport 80
--sport 80 : source port 80, on spécifit le port source du paquet
-j ACCEPT : on accepte de laisser le paquet

Maintenant, la réponse de notre requête HTTP revient sur notre parefeu et est acceptée !

12.2 Connexion Serveur sur notre parefeu



Par exemple, on fait tourner un serveur HTTP sur notre parefeu et l'on veut autoriser la connexion depuis l'internet. Cela signifie qu'un processus écoute sur le port 80.

Maintenant, la gymnastique des règles étant acquise, il paraît évident qu'il faut commencer par accepter les paquets en INPUT pour qu'elle arrive sur notre serveur Apache et qu'ensuite nous autorisions le paquet réponse à sortir de notre parefeu pour qu'elle soit envoyée au client :

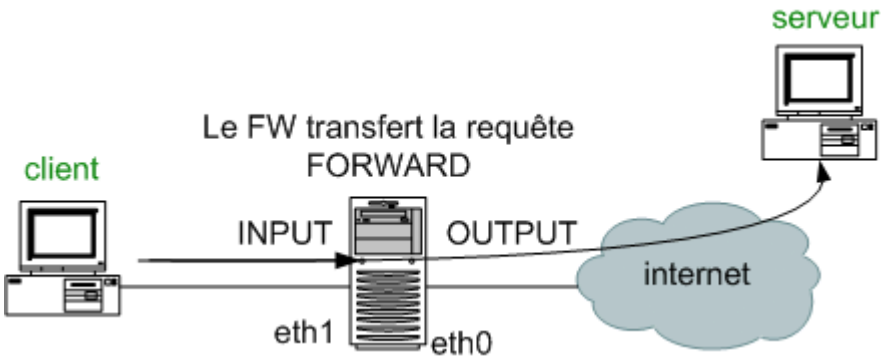
On autorise les paquets à destination de notre serveur WEB, c'est à dire les paquets pour port destination le port 80

```
# iptables -A INPUT -p TCP --dport 80 -j ACCEPT
```

On autorise les paquets ayant pour port source le port 80, c'est à dire les paquets réponses de notre serveur WEB, à sortir

```
# iptables -A OUTPUT -p TCP --sport 80 -j ACCEPT
```

12.3 Connexion cliente d'une machine du LAN



Une des machines du LAN veut se connecter sur "www.google.fr". Elle va envoyer la requête à sa passerelle, notre parefeu, qui va "relayer" sur Internet puis transmettre la réponse à notre machine. La requête va arriver sur l'interface "INSIDE" et doit être envoyé sur l'interface "OUTSIDE". Ce transfert s'appelle le "FORWARDING". La réponse revient et l'on doit "forwarder" de nouveau de l'interface "OUTSIDE" vers l'interface "INSIDE". La passerelle va relayer la requête et va donc être cliente du serveur Web "www.google.fr". On l'a vu précédement, voici ce que cela donne :

On autorise les paquets à destination d'un serveur WEB à passer de l'INSIDE vers l'OUTSIDE ou de l'OUTSIDE vers l'INSIDE (à savoir que de l'OUTSIDE vers l'INSIDE, il n'y a pas d'intérêt)

```
# iptables -A FORWARD -p TCP --dport 80 -j ACCEPT
```

On autorise les paquets ayant pour port source le port 80, c'est à dire les paquets réponses d'un serveur WEB, à passer de l'INSIDE vers l'OUTSIDE ou de l'OUTSIDE vers l'INSIDE

```
# iptables -A OUTPUT -p TCP --sport 80 -j ACCEPT
```

Ces deux règles sont très permissives... On pourrait ajouter des contraintes sur les adresses sources et cibles. Par exemple :

On autorise les paquets à destination d'un serveur WEB à passer de l'INSIDE vers l'OUTSIDE (et plus de l'outside vers l'inside)

```
# iptables -A FORWARD -p TCP -s 192.168.0.0/24 -d 0.0.0.0/0 --dport 80 -j
ACCEPT
```

On autorise les paquets ayant pour port source le port 80, c'est à dire les paquets réponses d'un serveur WEB, à passer de l'OUTSIDE vers l'INSIDE

```
# iptables -A FORWARD --p TCP -s 0.0.0.0/0 -d 192.168.0.0/24 --sport 80 -j
ACCEPT
```