

Fonctions réseau du noyau Linux

Laurent Foucher

laurent.foucher(at)iut-tlse3.fr

<http://www.linux-france.org/prj/inetdoc/>

Historique des versions		
\$Revision: 1.4 \$	\$Date: 2005/04/25 07:27:14 \$	
Année universitaire 2004-2005		

Configuration des fonctions réseau du système GNU/Linux.

Table des matières

1. Copyright et Licence	1
2. Présentation du noyau Linux	2
2.1. Généralités	2
2.1.1. Introduction	2
2.1.2. Architecture du noyau GNU/Linux	2
2.2. Les commandes sous Linux	4
2.3. Compilation du noyau	5
2.3.1. Construction du noyau	5
2.3.2. Utilisation des modules	5
3. Les technologies réseaux du noyau Linux	7
3.1. Les interfaces réseaux	7
3.1.1. ARCnet devices	7
3.1.2. Dummy net driver support	7
3.1.3. Bonding driver support	7
3.1.4. EQL (serial line load balancing) support	7
3.1.5. Universal TUN/TAP device driver support	8
3.1.6. General Instruments Surfboard 1000	9
3.1.7. Ethernet (10 or 100 Mbit)	9
3.1.8. Ethernet (1000 Mbit)	10
3.1.9. FDDI driver support	10
3.1.10. PLIP (parallel port) support	10
3.1.11. PPP (point-to-point protocol) support	11
3.1.12. SLIP (serial line) support	11
3.1.13. Wireless LAN (non-hamradio)	12
3.1.14. Token Ring devices	12
3.1.15. Fiber Channel driver support	12
3.1.16. Wan interfaces	12
3.2. Le sous-système RNIS	12
3.3. Les options réseaux	13
3.3.1. Packet Socket	13
3.3.2. Kernel/User netlink socket	14
3.3.3. Network packet filtering (replace ipchains)	14
3.3.4. Socket Filtering	14
3.3.5. Unix domain socket	14
3.3.6. TCP/IP networking	15
3.3.6.1. IP: multicasting	15
3.3.6.2. IP: advanced router	15
3.3.6.3. IP: kernel level autoconfiguration	16
3.3.6.4. IP: optimize as router not host	17
3.3.6.5. IP: tunneling	17
3.3.6.6. IP: GRE tunnel over IP	17
3.3.6.7. IP: TCP Explicit Congestion Notification support	17
3.3.6.8. IP: TCP syncookie support	17
3.3.6.9. IP: Allow large windows (not recommended if <16 Mb of memory)	17
3.3.6.10. IP: Netfilter configuration	17
3.3.7. 802.1Q VLAN Support	20
3.3.8. The IPX Protocol	20
3.3.9. Appletalk DDP	20
3.3.10. DECnet support	20
3.3.11. 802.1d Ethernet Bridging	21
4. Les outils réseaux du noyau Linux	22
4.1. Configuration des interfaces réseaux	22
4.1.1. ip link	22
4.1.2. ip address	22
4.1.3. ip rule	22
4.1.4. ip route	23
4.2. Sélection et contrôle des services	24

4.2.1. Le démon inetd	24
4.2.2. Le contrôle des services	24
4.3. Configuration du filtrage	25
4.3.1. Introduction	25
4.3.2. Architecture d'un firewall	25
4.3.3. Netfilter et iptables	25
4.3.4. Utilisation de l'outil iptables	25
4.3.5. Le filtrage avec iptables	26
4.3.6. Stateful, le suivi des connexions	27
4.3.7. La traduction d'adresse (NAT)	27

Liste des exemples

3.1. Exemple d'utilisation de l'interface dummy avec ifconfig	7
3.2. Exemple d'utilisation de l'interface dummy avec iproute2	7
3.3. Utilisation de la socket packet	8
3.4. Exemple d'utilisation de l'interface PLIP	10
3.5. Exemple d'utilisation du protocole PPP asynchrone	11
3.6. Exemple d'utilisation du protocole SLIP	11
3.7. Utilisation de la socket packet	13
3.8. Utilisation de la socket UNIX	14
3.9. Exemple de NAT	16
3.10. Exemple de chemins multiples	16
3.11. Exemple d'utilisation du champ TOS pour le routage	16
3.12. Exemple de configuration IP au démarrage	17
3.13. Limitation de consultation d'une règle	18
3.14. Filtrage de paquets en considérant l'adresse MAC	18
3.15. Filtrage de paquets par marquage de paquets	18
3.16. Exemple de filtrage basé sur la valeur du champ TOS	18
3.17. Filtrage de paquets en considérant l'adresse MAC	18
3.18. Exemple d'utilisation de unclean	19
3.19. Exemple de traduction d'adresse source	19
3.20. Exemple de traduction d'adresse de destination	19
3.21. Configuration du champ TOS	19
3.22. Log de paquets web	19
3.23. Exemple de configuration du support VLAN	20
4.1. contrôle d'accès	24
4.2. Autorisation d'une connexion ssh vers l'extérieur	27
4.3. SNAT standard	28
4.4. MASQUERADE	28

Chapitre 1. Copyright et Licence

Copyright (c) 2000,2005 Laurent Foucher.
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Copyright (c) 2000,2005 Laurent Foucher.
Permission est accordée de copier, distribuer et/ou modifier ce document selon les termes de la Licence de Documentation Libre GNU (GNU Free Documentation License), version 1.1 ou toute version ultérieure publiée par la Free Software Foundation ; sans Sections Invariables ; sans Texte de Première de Couverture, et sans Texte de Quatrième de Couverture. Une copie de la présente Licence est incluse dans la section intitulée « Licence de Documentation Libre GNU ».

Cet article est écrit avec *DocBook*¹ XML sur un système *Debian GNU/Linux*². Il est disponible en version imprimable aux formats PDF et Postscript : [interconnexion.fonctions.noyau.pdf](#)³ | [interconnexion.fonctions.noyau.ps.gz](#)⁴.

¹ <http://www.docbook.org>

² <http://www.debian.org>

³ <http://www.linux-france.org/prj/inetdoc/telechargement/interconnexion.fonctions.noyau.pdf>

⁴ <http://www.linux-france.org/prj/inetdoc/telechargement/interconnexion.fonctions.noyau.ps.gz>

Chapitre 2. Présentation du noyau Linux

Cette partie est une présentation très rapide de ce qu'est Linux et de ses principales caractéristiques.

2.1. Généralités

2.1.1. Introduction

Linux s'intègre dans la longue histoire des « Unix ». Le développement de ce système d'exploitation a débuté en 1969 sous l'impulsion de Ken Thompson et Dennis Ritchie qui travaillaient alors pour la société Bell Laboratories. Plusieurs versions furent développées en interne et c'est en 1975 qu'apparut la version 6 qui deviendra la base des Unix commerciaux. Par la suite, de nombreuses implémentations d'UNIX furent développées. L'université de Berkeley fut à la base de la version BSD, Hewlett Packard proposa la version HP-UX, etc... Malgré de bonnes intentions au départ, il existait des incompatibilités entre tous ces Unix, si bien que le portage d'une application d'un UNIX vers un autre était difficile. Pour réduire ces disparités, la société AT&T proposa un standard UNIX en 1983, connu sous le nom de System V. En 1986, l'Institute of Electrical and Electronics Engineers (IEEE) proposa un autre standard connu sous le terme de POSIX. POSIX est une standardisation permettant d'assurer la portabilité des applications d'un UNIX à un autre.

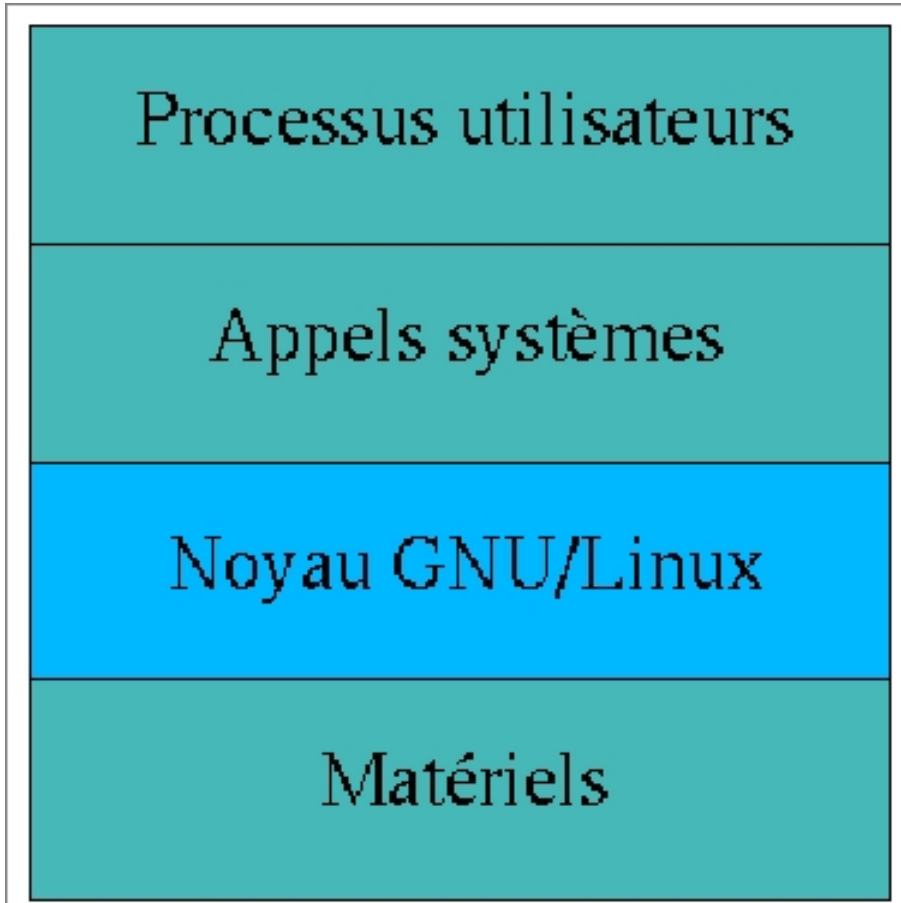
Linux est un système d'exploitation qui se comporte comme un UNIX et qui implémente les spécifications POSIX, avec des extensions système V et Berkeley. Issu du travail d'un étudiant finlandais, Linus Torvalds, Linux se distingue par le fait qu'il est distribué sous les conditions d'une licence particulière, appelée GPL (GNU Public License). Cette licence précise que toute personne peut modifier, améliorer ou corriger le code source, mais que ces modifications devront également être distribuées librement. Les principales caractéristiques de Linux sont les suivantes :

- Multitâches : exécute plusieurs programmes en même temps
- Multi-utilisateurs : plusieurs utilisateurs peuvent être actifs en même temps
- Multi plates-formes : Linux peut fonctionner avec différents types de processeurs (Intel, Sparc, Alpha, PowerPC, etc...)
- Supporte un grand nombre de systèmes de fichiers : Ext2, MSDOS, FAT, VFAT, etc....
- Dispose d'une partie réseau conséquente. Voir [Chapitre 3, Les technologies réseaux du noyau Linux](#)

L'évolution du noyau est très rapide et celui qui est utilisé pour ce programme de formation appartient à la série 2.4.

2.1.2. Architecture du noyau GNU/Linux

Comme tout système d'exploitation, le noyau Linux est une interface entre des programmes utilisateurs et des périphériques physiques. L'accès à ces périphériques se fait par l'intermédiaire d'appels systèmes qui sont identiques quelle que soit la machine. Cette encapsulation du matériel libère les développeurs de logiciels de la gestion complexe des périphériques : c'est le système d'exploitation qui s'en charge. Ainsi, si le système d'exploitation existe sur plusieurs architectures, l'interface d'utilisation et de programmation sera la même sur toutes. On dira alors que le système d'exploitation offre une *machine virtuelle* à l'utilisateur et aux programmes qu'il exécute.



GNU/Linux est considéré comme un système d'exploitation monolithique, écrit comme un ensemble de procédures qui peuvent s'appeler mutuellement. Pour l'utilisateur, il se présente comme un seul gros fichier. Cependant, il contient un ensemble de composants réalisant chacun une tâche bien précise. Cette construction monolithique induit un aspect important : la notion *d'espace noyau* et *d'espace utilisateur*. Dans l'espace noyau, aucune restriction n'est imposée. Dans l'espace utilisateur, un certain nombre de restrictions sont imposées (par exemple, la création d'un fichier ne peut se réaliser que si les droits sont suffisants), et le processus ne peut avoir accès qu'aux zones mémoires qui lui ont été allouées.

Le noyau GNU/Linux est composé de cinq sous-systèmes principaux. Un sous-système peut être défini comme une entité logicielle qui fournit une fonctionnalité particulière.

Tâches réalisées par GNU/Linux

Gestion des processus

Ce sous-système est chargé de répartir équitablement les accès au processeur entre toutes les applications actives. Cela n'inclut pas seulement les processus utilisateurs, mais aussi les sous-systèmes du noyau lui-même. Cette fonction est réalisée par le *scheduler*.

Gestion de la mémoire

Ce sous-système est chargé d'affecter à chaque programme une zone mémoire. Il a également un rôle de protection : la mémoire pour un processus est privée et celle-ci ne doit pas être lue ni modifiée par un autre.

Système de fichier virtuel

Le sous-système de fichiers garantit une gestion correcte des fichiers et un contrôle des droits d'accès. Pour limiter la complexité liée aux nombreux systèmes de fichiers existants, Linux adopte le concept de *Virtual FileSystem* (VFS). Le principe de VFS est de proposer des appels systèmes identiques quel que soit le système de fichiers. Il est de la responsabilité du noyau de détourner les appels standards vers les appels spécifiques au système de fichiers.

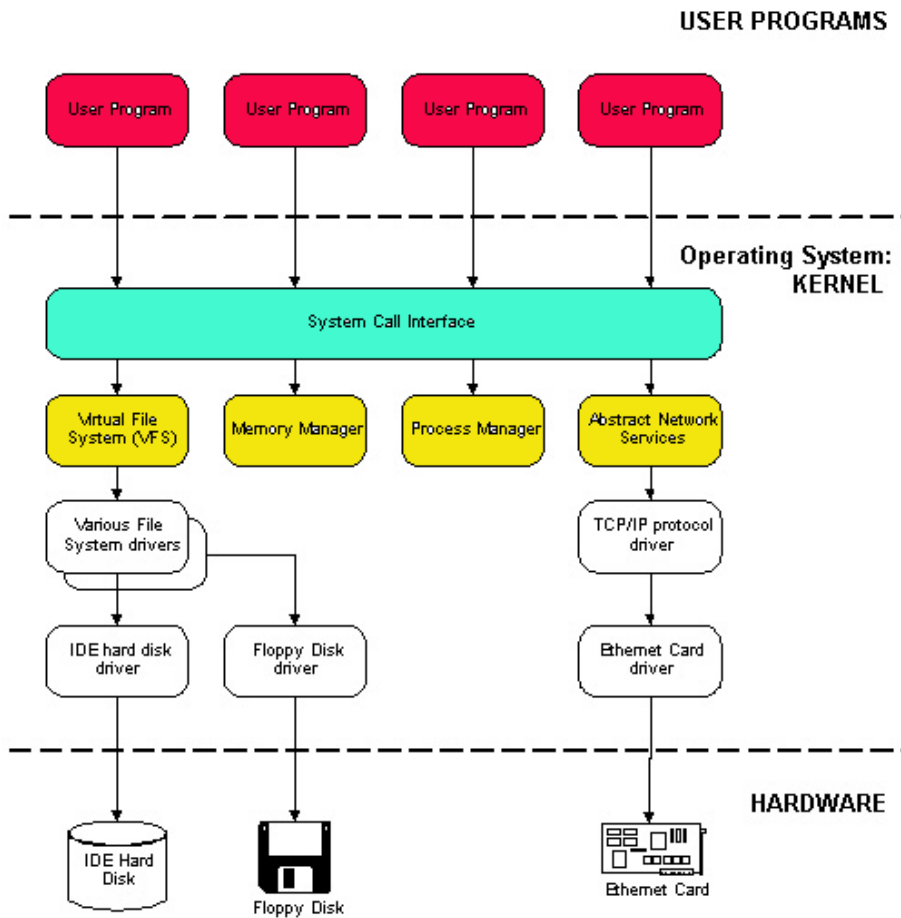
Service réseau

Le sous-système réseau permet à Linux de se connecter à d'autres systèmes à travers un réseau informatique. Il y a de nombreux périphériques matériels qui sont supportés et plusieurs protocoles réseaux peuvent être utilisés.

Communications Inter Processus

Dans la mesure où un processus ne peut avoir accès qu'à la zone mémoire qui lui a été allouée, Linux propose plusieurs mécanismes permettant à des applications de communiquer entre elles.

Les relations entre les différentes parties du noyau sont montrées sur la figure ci-dessous.



2.2. Les commandes sous Linux

Même si la partie graphique de Linux prend de plus en plus d'importance, il est intéressant de connaître quelques commandes en mode texte. Pour pouvoir utiliser ces commandes, il convient de disposer d'un terminal. En mode texte, celui-ci est automatiquement disponible, alors qu'en mode graphique, il faut un émulateur de terminal. Le plus connu est certainement xterm. La syntaxe de base pour une commande UNIX est la suivante :

```
commande [-options] [arg1] [arg2...]
```

Les options et le nombre d'arguments dépendent de la fonction. Pour obtenir des informations sur l'utilisation d'une commande, on peut consulter les pages de manuels. On obtient ces informations grâce à la commande **man nom_commande**.

Voici une petite liste de commandes susceptibles d'être utilisées :

Liste de commandes usuelles

passwd

Modifier son mot de passe

date

Afficher la date et l'heure

cat

Concaténer des fichiers

id

Fournit des informations sur l'utilisateur

Un guide très complet existe et est disponible [ici](#)¹

2.3. Compilation du noyau

2.3.1. Construction du noyau

Le noyau Linux propose à l'heure actuelle la gestion de nombreux matériels qui ne sont pas forcément présents sur la machine. Compiler le noyau, c'est choisir les pilotes des composants matériels qui devront être intégrés et ainsi l'adapter à l'ordinateur.

Les sources du noyau Linux se trouvent dans le répertoire `/usr/src/linux`. La première étape consiste à définir les options à inclure dans le noyau. Pour cela, il existe trois possibilités. Il suffit de lancer l'une des commandes suivantes :

- **make config**
- **make menuconfig**
- **make xconfig**

Une fois la configuration sauvegardée, il faut lancer la compilation proprement dite avec la commande **make dep ; make clean ; make bzImage**

Pour démarrer avec le nouveau noyau, il faut le copier vers le répertoire de boot :

```
#cp arch/i386/boot/bzImage  
/boot/vmlinuz-X.X.X
```

où X.X.X est un numéro de version.

Enfin, il faut faire reconnaître le noyau au chargeur, qui est, dans le cas général, LILO. Pour cela, on ajoute au fichier `/etc/lilo.conf` les lignes suivantes :

```
image=/boot/vmlinuz-X.X.X  
label=Linux  
read-only
```

Il ne faut pas oublier de relancer lilo avec la commande `/sbin/lilo`.

2.3.2. Utilisation des modules

La frontière espace noyau/espace utilisateur du noyau GNU/Linux pose un problème : ajouter de nouvelles fonctionnalités implique d'étendre le noyau lui-même. Dans les anciennes versions du noyau, il fallait, par exemple, recompiler l'ensemble du noyau à chaque ajout d'un matériel. Les modules sont apparus depuis la version 2.0 et cette fonctionnalité permet de corriger ce défaut. Ces modules correspondent à des pilotes que l'on peut charger et décharger dynamiquement en mémoire. Pour pouvoir bénéficier des modules, le noyau doit être obligatoirement compilé avec l'option « module » activée.

Pour créer un pilote en tant que module, il suffit de préciser l'option `M` lors du choix des options du noyau. La compilation se réalise grâce à la commande **make modules**. Une fois les modules compilés, ceux-ci sont placés dans le répertoire `/lib/modules/(version noyau)` grâce à la commande **make modules_install**. D'autres sous-répertoires seront créés en fonction du choix des options. Par exemple, le pilote Hisax se trouvera dans le sous-répertoire `misc`.

Une fois que les modules sont compilés et installés, on peut les manipuler grâce aux commandes suivantes :

- `lsmod` : permet de lister l'ensemble des modules présents en mémoire
- `insmod` : permet d'insérer un module en mémoire.
- `rmmmod` : permet de supprimer un module de la mémoire.
- `depmod` : permet la création d'une liste des modules utilisables avec le noyau.

¹ <http://rute.sourceforge.net>

- modprobe : permet d'insérer un module en mémoire, ainsi que toutes ses dépendances.

Chapitre 3. Les technologies réseaux du noyau Linux

Le code des technologies réseaux intégrées dans le noyau est généralement appelé : *kernel space code*.

3.1. Les interfaces réseaux

Network device support

Afin de pouvoir mettre en place un réseau sous Linux, il faut bien évidemment que les interfaces matérielles soient reconnues. L'ensemble des cartes réseaux supportées par Linux est regroupé dans l'option nommée *Network device support*, tandis que les cartes RNIS sont référencées dans l'option *ISDN subsystem*. Dans cette partie de la configuration, on s'intéresse au matériel, mais aussi à certains protocoles de la couche 2 du modèle OSI.

3.1.1. ARCnet devices

Arcnet un type de réseau comparable à Ethernet, mais qui n'est plus guère utilisé.

3.1.2. Dummy net driver support

Pour une machine isolée, sans matériel réseau spécifique, la seule interface réseau active est l'interface de boucle locale (*loopback*), qui a normalement l'adresse 127.0.0.1. Cette interface *dummy* permet de spécifier une autre adresse IP officielle pour la machine. Par exemple, une machine peut avoir son nom résolu avec l'adresse IP 192.168.0.1. Si la machine n'a pas d'interfaces physiques, le noyau ne sait pas que cette adresse IP référence également cette machine et le datagramme envoyé à 192.168.0.1 est éliminé. L'interface *dummy* résout ce problème. Il suffit d'attribuer l'adresse 192.168.0.1 à l'interface *dummy*.

Exemple 3.1. Exemple d'utilisation de l'interface dummy avec ifconfig

```
$ ifconfig dummy0 192.168.2.1 netmask 255.255.255.0
$ ifconfig dummy0

dummy0    Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:192.168.2.1  Bcast:192.168.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

Exemple 3.2. Exemple d'utilisation de l'interface dummy avec iproute2

```
$ip addr add 192.168.2.1/24 brd + dev dummy0
$ip addr ls dev dummy0

8: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop
  link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
  inet 192.168.2.1/24 brd 192.168.2.255 scope global dummy0
```

3.1.3. Bonding driver support

Cette option permet d'assembler les transmissions de plusieurs interfaces Ethernet en une seule connexion virtuelle. Il y a alors agrégation de la bande passante. Les canaux de transmission utilisant cette technique sont aussi appelés *trunks*.

3.1.4. EQL (serial line load balancing) support

Permet de partager le flux TCP/IP à travers deux interfaces séries. Identique à l'option précédente dans le principe.

3.1.5. Universal TUN/TAP device driver support

TUN/TAP offre la possibilité à des programmes utilisateurs de transmettre et recevoir des paquets. Cela peut être vu comme un périphérique point à point (TUN) ou Ethernet (TAP) qui au lieu de recevoir des paquets d'un média physique, les reçoit d'un programme utilisateur. L'application principale est le tunneling.

Exemple 3.3. Utilisation de la socket packet

```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <linux/if.h>
#include <linux/if_tun.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <errno.h>
#include <unistd.h>

u_int16_t csum_partial(void *buffer, unsigned int len, u_int16_t prevsum)
{
    u_int32_t sum = 0;
    u_int16_t *ptr = buffer;

    while (len > 1) {
        sum += *ptr++;
        len -= 2;
    }
    if (len) {
        union {
            u_int8_t byte;
            u_int16_t wyde;
        } odd;
        odd.wyde = 0;
        odd.byte = *((u_int8_t *)ptr);
        sum += odd.wyde;
    }
    sum = (sum >> 16) + (sum & 0xFFFF);
    sum += prevsum;
    return (sum + (sum >> 16));
}

int main ()
{
    struct ifreq ifr;
    int fd, len, err;
    char *buf ;
    char *dev="";
    union {
        struct {
            struct iphdr ip;
        } fmt;
        unsigned char raw[65536];
    } u;

    if( (fd = open("/dev/net/tun", O_RDWR)) < 0 ) {
        printf("Impossible d'ouvrir /dev/net/tun\n");
        return -1;
    }

    memset(&ifr, 0, sizeof(ifr));

    /* Flags: IFF_TUN   - TUN device (no Ethernet headers)
     *         IFF_TAP   - TAP device
     *
     *         IFF_NO_PI - Do not provide packet information
     */
    ifr.ifr_flags = IFF_TUN|IFF_NO_PI ;
    if( *dev )
        strncpy(ifr.ifr_name, dev, IFNAMSIZ);

    if( (err = ioctl(fd, TUNSETIFF, (void *) &ifr)) < 0 ){
        close(fd);
        return -1;
    }
    printf("le device utilisé est : %s \n",ifr.ifr_name);
}
```

```

while ((len = read(fd, &u, sizeof(u))) > 0) {
    u_int32_t tmp;
    struct icmp_hdr *icmp
        = (void *)((u_int32_t *)&u.fmt.ip + u.fmt.ip.ihl);
    struct tcp_hdr *tcp = (void *)icmp;
    struct udp_hdr *udp = (void *)icmp;

    fprintf(stderr, "SRC = %u.%u.%u.%u DST = %u.%u.%u.%u\n",
        (ntohl(u.fmt.ip.saddr) >> 24) & 0xFF,
        (ntohl(u.fmt.ip.saddr) >> 16) & 0xFF,
        (ntohl(u.fmt.ip.saddr) >> 8) & 0xFF,
        (ntohl(u.fmt.ip.saddr) >> 0) & 0xFF,
        (ntohl(u.fmt.ip.daddr) >> 24) & 0xFF,
        (ntohl(u.fmt.ip.daddr) >> 16) & 0xFF,
        (ntohl(u.fmt.ip.daddr) >> 8) & 0xFF,
        (ntohl(u.fmt.ip.daddr) >> 0) & 0xFF);

    switch (u.fmt.ip.protocol) {
    case IPPROTO_ICMP:
        if (icmp->type == ICMP_ECHO) {
            fprintf(stderr, "PONG! (iphdr = %u bytes)\n",
                (unsigned int)((char *)icmp
                    - (char *)&u.fmt.ip));

            /* Turn it around */
            tmp = u.fmt.ip.saddr;
            u.fmt.ip.saddr = u.fmt.ip.daddr;
            u.fmt.ip.daddr = tmp;

            icmp->type = ICMP_ECHOREPLY;
            icmp->checksum = 0;
            icmp->checksum
                = ~csum_partial(icmp,
                    ntohs(u.fmt.ip.tot_len)
                    - u.fmt.ip.ihl*4, 0);

            {
                unsigned int i;
                for (i = 44;
                    i < ntohs(u.fmt.ip.tot_len); i++){
                    printf("%u:0x%02X ", i,
                        ((unsigned char *)
                            &u.fmt.ip)[i]);
                }
                printf("\n");
            }
            write(fd, &u, len);
        }
        break;
    case IPPROTO_TCP:
        fprintf(stderr, "TCP: %u -> %u\n", ntohs(tcp->source),
            ntohs(tcp->dest));
        break;
    case IPPROTO_UDP:
        fprintf(stderr, "UDP: %u -> %u\n", ntohs(udp->source),
            ntohs(udp->dest));
        break;
    }
}
return 0;
}

```

3.1.6. General Instruments Surfboard 1000

Permet le support des modems-câbles de la société General Instruments.

3.1.7. Ethernet (10 or 100 Mbit)

Ce menu regroupe l'ensemble des cartes ethernet 10 et 100 Mbits supportées par Linux. Les principales cartes supportées sont les suivantes :

- cartes 3COM 3C5x, 3C59c et 3C900

- cartes de la marque Western digital
- cartes de la marque Racal
- cartes PCMCIA pour portables

3.1.8. Ethernet (1000 Mbit)

Drivers pour des cartes ethernet 1000 Mbits.

3.1.9. FDDI driver support

FDDI désigne un réseau LAN en boucle à base de fibre optique. Ce type de réseau offre un débit de 100 Mbits/s sur des distances de raccordement pouvant atteindre 200 km. FDDI est un réseau de type 802 et peut être utilisé pour la création d'un réseau LAN. Cependant, vu le débit élevé qu'il propose, celui-ci est plutôt destiné à jouer un rôle fédérateur de réseau LAN à plus faible débit.



Note sur les réseaux de type 802

Les réseaux de type 802 sont des réseaux locaux qui suivent la normalisation 802 de l'IEEE (Institute of Electrical & Electronic Engineering). Cette norme introduit une division de la couche Liaison du modèle OSI en deux sous-couches. La sous-couche LLC (Logical Link Control) offre une interface unique pour la couche réseau quel que soit le support physique. La sous-couche MAC (Medium Access Control) définit le mode d'accès au média (CSMA/CD, bus à jeton, anneau à jeton, etc...).

3.1.10. PLIP (parallel port) support

Ce périphérique permet la mise en place d'un réseau point à point en utilisant le port parallèle.

Exemple 3.4. Exemple d'utilisation de l'interface PLIP

Deux machines (MachineA et MachineB) sont reliées par un câble *LAP-LINK* sur leur port parallèle.

```
Sur la MachineA et la MachineB :
$ modprobe parport_pc io=0x378 irq=7

Sur la MachineA :
$ ifconfig plip0 192.168.3.1 pointopoint 192.168.3.2
$ ifconfig plip0

plip0      Link encap:Ethernet  HWaddr FC:FC:C0:A8:03:01
           inet addr:192.168.3.1  P-t-P:192.168.3.2  Mask:255.255.255.255
           UP POINTOPOINT RUNNING NOARP  MTU:1500  Metric:1
           RX packets:0 errors:0 dropped:0 overruns:0 frame:0
           TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:10
           Interrupt:7 Base address:0x378

Sur la MachineB :
$ ifconfig plip0 192.168.3.2 pointopoint 192.168.3.1

Ou l'équivalent avec iproute2 :

$ ip addr add 192.168.3.1/32 peer 192.168.3.2/32 dev plip0
$ ip addr ls dev plip0

5: plip0: <POINTOPOINT,NOARP> mtu 1500 qdisc pfifo_fast qlen 10
   link/ether fc:fc:c0:a8:03:01 peer ff:ff:ff:ff:ff:ff
   inet 192.168.3.1 peer 192.168.3.2/32 scope global plip0
```

Pour plus de détails, voir le fichier `Documentation/networking/parport.txt` dans les sources du noyau.

3.1.11. PPP (point-to-point protocol) support

PPP est un protocole de la couche 2 du modèle OSI qui permet de relier deux machines par l'intermédiaire du port série. PPP gère la détection des erreurs, permet la négociation des adresses IP à la connexion, ainsi que l'authentification.

Sous-option de PPP

PPP multilink support

Cette option permet d'utiliser un mécanisme d'entrée-sortie plus rapide.

PPP support for async serial port

Cette option permet d'utiliser PPP sur les ports séries asynchrones. Cette option est nécessaire pour une connexion à un fournisseur d'accès Internet via un modem analogique. Ce mécanisme de connexion peut également être utilisé pour réaliser une connexion point à point entre deux machines reliées par un câble *Null Modem*.

Exemple 3.5. Exemple d'utilisation du protocole PPP asynchrone

Deux machines (MachineA et MachineB) sont reliées par un câble *NULL Modem*.

```

Sur la MachineA :
$ pppd /dev/ttyS0 115200 crtscts noauth 192.168.2.1: &

Sur la MachineB :
$ pppd /dev/ttyS0 115200 crtscts noauth 192.168.2.2: &
$ ifconfig ppp0

ppp0      Link encap:Point-to-Point Protocol
          inet addr:192.168.2.2  P-t-P:192.168.2.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:53 errors:0 dropped:0 overruns:0 frame:0
          TX packets:58 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3

```

PPP support for sync tty port

Permet le support PPP sur des lignes synchrones, utilisant par exemple le protocole HDLC.

PPP Deflate compression

Cette option permet d'activer la compression des paquets PPP avant leur envoi sur la ligne. L'algorithme de compression utilisé ici est le même que celui utilisé par gzip (Deflate Algorithm).

PPP BSD-Compress compression

Compression des paquets PPP grâce à la méthode de compression BSD-Compress.

PPP over ethernet

Le protocole PPPoE est utilisé par certains fournisseurs d'accès ADSL. Pour plus d'informations sur ce sujet, consulter la page [roaringpenguin](http://www.roaringpenguin.com/pppoe/)¹ pour un exemple de connexion.

3.1.12. SLIP (serial line) support

SLIP est le premier protocole qui a permis d'envoyer du trafic IP sur des connexions séries, comme les lignes téléphoniques ou les câbles *Null Modem*. Les fonctionnalités de ce protocole sont très réduites. Il est principalement employé pour accéder à la configuration des châssis d'interconnexion en mode texte.

Exemple 3.6. Exemple d'utilisation du protocole SLIP

Deux machines (MachineA et MachineB) sont reliées par un câble *NULL Modem*.

```

Sur la MachineA :

```

¹ www.roaringpenguin.com/pppoe/

```

$ slattach -p slip /dev/ttyS1 &
$ ifconfig s10 192.168.2.1 pointopoint 192.168.2.2

Sur la MachineB :
$ slattach -p slip /dev/ttyS0 &
$ ifconfig s10 192.168.2.2 pointopoint 192.168.2.1
$ ifconfig s10

s10      Link encap:Serial Line IP
        inet addr:192.168.2.2  P-t-P:192.168.2.1  Mask:255.255.255.255
        UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:296  Metric:1
        RX packets:2 errors:0 dropped:0 overruns:0 frame:0
        TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:10

```

3.1.13. Wireless LAN (non-hamradio)

Permet le support du matériel sans fil pour le transport des données. Cela permet en particulier le transfert de données par ondes radios.

3.1.14. Token Ring devices

Drivers pour les cartes Token Ring. Token Ring est un protocole de la couche 2 du modèle OSI et est soumis à la norme IEEE 802.5. Dans cette technologie, la principale différence avec le support Ethernet est la méthode d'accès au média physique. En effet, alors que pour Ethernet la méthode d'accès se fait par l'écoute du média, une station d'un réseau Token Ring ne pourra émettre que si elle est en possession d'un jeton virtuel. En effet, sur un anneau à jeton, une séquence binaire particulière appelée jeton (token) circule en permanence lorsque les stations n'ont rien à transmettre. Lorsqu'une station veut émettre une trame, elle doit prendre possession du jeton et le remplacer par la trame qu'elle souhaite transmettre. Comme il n'y a qu'un seul jeton en circulation sur l'anneau, seule une station peut émettre à un instant donné, ce qui évite l'émission simultanée de plusieurs trames.

3.1.15. Fiber Channel driver support

Ce réseau est basé sur un support par fibres optiques et peut atteindre un débit de 100, 200, 400 et 800 Mbits/s. Cette technologie peut être vue comme un canal d'ordinateur ou comme un réseau de communication. Elle peut notamment être utilisée pour connecter de grands périphériques de stockage à un ordinateur. En effet, un réseau Fiber Channel peut acheminer par encapsulation des données issues des canaux SCSI, mais également les données des réseaux 802, FDDI, IP ou ATM.

3.1.16. Wan interfaces

Dans cette option, on trouvera du matériel permettant d'interconnecter des réseaux locaux pour réaliser des réseaux étendus (WAN). On trouvera des cartes qui permettent le dialogue à travers des lignes séries, à travers le protocole X25 ou à travers le protocole Frame-Relay.

3.2. Le sous-système RNIS

ISDN subsystem

Cette rubrique rassemble les options des connexions RNIS, ainsi que les pilotes de cartes. Ces options sont décrites dans la [3ème partie](#)² du *Guide RNIS Linux*.

Voici une liste des options utiles pour nos applications :

- ISDN support->m
- support synchronous PPP->y
 - Use VJ-compression with synchronous PPP->y
 - Support generic MP (RFC 1717)->y
- ISDN feature submodules : isdnloop support->m

² <http://www.linux-france.org/prj/inetdoc/guides/rnis/part3.html>

- Passive ISDN cards : HiSax Siemens ChipSet driver support->m
 - HiSax Support for EURO/DSS1->y
 - HiSax Supported cards : Gazel cards->y

3.3. Les options réseaux

Networking options

Le but de cette section est de découvrir les diverses options réseaux que propose le noyau Linux. Ici, on s'intéresse aux couches réseaux à partir de la couche 3.

3.3.1. Packet Socket

Cette fonctionnalité est utilisée pour recevoir ou envoyer des paquets bruts sur les périphériques réseaux sans passer par l'intermédiaire d'un protocole réseau implémenté dans le noyau. Certains programmes, tel que tcpdump, utilisent cette option.

Le terme « socket » désigne l'interface de programmation à travers laquelle l'on va pouvoir accéder aux ressources réseaux du noyau. La création d'une socket est réalisée par l'appel système suivant :

```
int socket(famille, type, protocole);
int famille;
int type;
int protocole;
```

Le paramètre « famille » permet de préciser avec quel protocole réseau l'on souhaite travailler. L'ensemble des familles disponibles est listé dans le fichier `/usr/include/linux/socket.h`. Les types définissent le mode de connection (TCP ou UDP). Une nouvelle famille de socket associée à cette fonctionnalité est ainsi disponible, à savoir `AF_PACKET`.

Sous-option de Packet Socket

mapped IO

Cette option permet d'utiliser un mécanisme d'entrée-sortie plus rapide.

Exemple 3.7. Utilisation de la socket packet

```
#include <stdio.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <linux/if_ether.h>
#include <linux/if_packet.h>

main ()
{
    int sock_fd ;
    struct sockaddr_ll sll;
    struct ifreq ifr;
    char buffer[2000];
    int nb_octet;

    if (sock_fd = socket(AF_PACKET,SOCK_RAW,htons(ETH_P_ALL)) == -1 ) {
        printf("Erreur dans la création de la socket\n");
        return -1 ;
    }

    memset(&ifr, 0, sizeof(ifr));
    strncpy (ifr.ifr_name, "eth0", sizeof(ifr.ifr_name));

    if (ioctl(sock_fd,SIOCGIFINDEX, &ifr) == -1 ) {
        printf("Erreur dans la recherche de index\n");
        return -1 ;
    }

    memset(&sll, 0, sizeof(sll));
    sll.sll_family = AF_PACKET ;
```

```

sll.sll_ifindex = ifr.ifr_ifindex ;
sll.sll_protocol = htons(ETH_P_ALL);

if (bind(sock_fd, (struct sockaddr *) &sll, sizeof(sll)) == -1) {
    printf("Erreur avec bind\n");
    return -1 ;
};

nb_octet=recvfrom(sock_fd,buffer,sizeof(buffer),0,NULL,0);
printf("Nombre d'octets reçus : %d\n",nb_octet);
}

```

3.3.2. Kernel/User netlink socket

Kernel/User netlink socket : Définit une nouvelle famille de socket, AF_NETLINK. Cette socket permet d'établir une communication bidirectionnelle entre le noyau et l'espace utilisateur. Cette option est nécessaire pour pouvoir utiliser l'outil iproute2 qui permet la configuration de la partie réseau du noyau. En plus de cette socket, la communication peut également se réaliser, pour un processus utilisateur, par la lecture ou l'écriture de fichiers caractères spéciaux. Ces fichiers spéciaux ont le numéro majeur 36 et se trouvent dans le répertoire /dev.

Sous-option de netlink

Routing Messages

Le noyau fournit des informations sur le routage via le fichier /dev/route de numéro majeur 36 et de numéro mineur 0.

Netlink Device Emulation

Permet la compatibilité avec d'anciennes fonctionnalités. Option amenée à disparaître.

3.3.3. Network packet filtering (replace ipchains)

Cette option active la fonction de filtrage des paquets traversant la machine Linux. Le filtrage permet un blocage sélectif du trafic IP en fonction, par exemple, de l'origine ou de la destination.

Sous-option de Network packet filtering

Network packet filtering debugging

Permet d'avoir des messages supplémentaires du code netfilter.

3.3.4. Socket Filtering

Cette fonctionnalité permet, dans les programmes en mode utilisateur, la mise en place de filtres au niveau des sockets. On a ainsi la possibilité d'autoriser ou d'interdire des types de données traversant une socket. Cette fonctionnalité est dérivée du filtrage de paquets Berkeley. Pour plus d'informations, voir le fichier Documentation/networking/filter.txt dans les sources du noyau.

3.3.5. Unix domain socket

Permet la prise en charge des sockets du domaine UNIX. X-windows et syslog sont des exemples de programmes qui utilisent ce type de fonctionnalité. Les sockets UNIX ne permettent que des communications locales sur une machine. Ce type de socket est lié à la création d'un fichier. Le nom de la famille associé aux sockets du domaine Unix est AF_UNIX.

Exemple 3.8. Utilisation de la socket UNIX

```

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>

```

```

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>

int main(void)
{
    int socket_unix, len;
    struct sockaddr_un local;

    if ((socket_unix = socket(AF_UNIX, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    local.sun_family = AF_UNIX;
    strcpy(local.sun_path, "/tmp/test_socket_unix");
    unlink(local.sun_path);
    len = strlen(local.sun_path) + sizeof(local.sun_family);

    if (bind(socket_unix, (struct sockaddr *)&local, len) == -1) {
        perror("bind");
        exit(1);
    }

    system("ls -l /tmp/");

    unlink(local.sun_path);
    return 0;
}

```

3.3.6. TCP/IP networking

Active le protocole TCP/IP.

3.3.6.1. IP: multicasting

Permet d'envoyer des paquets à plusieurs ordinateurs en même temps. Cette fonctionnalité est, par exemple, utilisée pour la diffusion audio et vidéo.

3.3.6.2. IP: advanced router

Par défaut, la décision du routage se fait en examinant l'adresse de destination. En activant cette option, on peut contrôler beaucoup plus précisément le routage et la prise de décision pourra se faire en fonction de nombreux autres critères.

Sous-option de advanced router

policy routing

Permet le remplacement de la table de routage classique, basée sur les adresses de destination, par la Base de Données des Politiques de Routage ou Routing Policy DataBase (RPDB) en anglais. Cette base de données est une liste ordonnée de règles qui scrutent certains caractéristiques des paquets :

- adresse source
- adresse de destination
- champ TOS
- marque du packet
- interface d'entrée

Si un paquet satisfait les spécifications d'une règle, alors l'action correspondante est réalisée. L'action standard consiste à fournir l'adresse IP du prochain saut. Si vous souhaitez plus d'informations sur ce sujet, je vous conseille la lecture de deux documents :

- La documentation ip-cref d'Alexey Kuznetsov, disponible dans les sources de l'utilitaire iproute2

- L'article "[Policy Routing in Linux](#)"³.

De plus, un exemple complet de routage avancé pourra être consulté dans le [Linux 2.4 Advanced routing Howto](#)⁴.

Si l'option "IP: use netfilter MARK value as routing key" est validée, le routage des paquets pourra s'établir en fonction de la marque du paquet. Un exemple pourra être consulté dans le [Linux 2.4 Advanced routing Howto](#)⁵.

Si l'option "IP: fast network translation adress" est validée, le routeur pourra modifier les adresses source et destination des paquets transmis.

Exemple 3.9. Exemple de NAT

Soit un routeur avec d'un coté un réseau local 192.168.1.0/24 et de l'autre un réseau public (200.200.200.0/24, par ex) ayant une connectivité sur Internet. On souhaite qu'une machine du réseau local (192.168.1.1, par exemple) soit reconnu avec l'adresse 200.200.200.10 sur Internet.

```
$ ip route add nat 200.200.200.10 via 192.168.1.1
$ ip rule add prio 300 from 192.168.1.1 nat 200.200.200.10
```

equal cost multipath

Avec cette option, on peut spécifier plusieurs routes alternatives que peuvent emprunter les paquets. Le routeur considère toutes ces routes comme étant de coûts égaux et choisit l'une d'elle d'une manière non déterministe si un paquet qui correspond arrive.

Exemple 3.10. Exemple de chemins multiples

Considérons un routeur avec deux liaisons ppp. On souhaite que les paquets sortant puissent utiliser indifféremment ppp0 ou ppp1 comme interface de route par défaut.

```
$ ip route add default scope global nexthop dev ppp0 nexthop dev ppp1
```

use TOS value as routing key

L'entête d'un paquet IP contient un champ de 8 bits nommé Type Of Service (Type de service). Dans ce champ, il y a trois indicateurs qui permettent de préciser le type d'acheminement souhaité : Délai faible (faible temps d'attente), débit important et fiabilité importante. Cela permet de choisir entre, par exemple, une liaison satellite à haut débit mais avec un délai d'attente important ou une ligne louée à faible débit et faible délai. Cette option permet d'utiliser la valeur du champ TOS dans la liste de règle.

Exemple 3.11. Exemple d'utilisation du champ TOS pour le routage

But : tous les paquets marqués avec le champ TOS "débit important" (0x08) (par exemple le transfert de données via ftp) doivent empruntés une liaison RNIS.

```
$ ip rule add tos 0x08 prio 100 table 10
$ ip route add default dev ippp0 table 10
```

verbose route monitoring

Permet l'affichage de messages au sujet du routage.

large routing tables

Si la table de routage possède plus de 64 entrées, il est préférable d'activer cette option pour accélérer le processus de routage.

3.3.6.3. IP: kernel level autoconfiguration

Cette option permet de configurer les adresses IP des périphériques au moment du démarrage, ainsi que la table de

³ <http://www.samag.com/linux/articles/v09/i01/a3.htm>

⁴ <http://www.linux-france.org/prj/inetdoc/i/net/guides/Advanced-routing-Howto/Advanced-routing-Howto.v130-4.html>

⁵ <http://www.linux-france.org/prj/inetdoc/i/net/guides/Advanced-routing-Howto/Advanced-routing-Howto.v130-12.html>

roulage. Les informations nécessaires à cette configuration sont fournies soit sur la ligne de commande, soit par l'intermédiaire des protocoles DHCP, BOOTP ou RARP.

Les informations sont fournies au noyau via le paramètre `ip`. Cette option est principalement utilisée pour la mise en place de stations clientes sans disque dur et qui ont besoin de monter la racine du système de fichiers par nfs. Pour plus d'informations, voir le fichier `Documentation/nfsroot.txt` dans les sources du noyau.

Exemple 3.12. Exemple de configuration IP au démarrage

```
LILO: linux ip=192.168.1.1::192.168.1.254:255.255.255.0:Linuxbox:eth0:none
```

3.3.6.4. IP: optimize as router not host

Permet de supprimer certaines vérifications lorsque le noyau reçoit un paquet. Dans le cas où Linux est principalement utilisé comme un routeur, c'est-à-dire une machine qui ne fait que transmettre les paquets, cela permet d'améliorer la vitesse.

3.3.6.5. IP: tunneling

Le tunneling permet l'encapsulation d'un protocole réseau dans un autre protocole réseau. Cette option permet l'encapsulation du protocole IP dans IP. Cela peut être utilisé dans le cas où l'on souhaite pouvoir faire communiquer deux réseaux ayant des adresses privées, donc non routables, à travers Internet. Un exemple complet de tunnel IP dans IP pourra être consulté dans le [Linux 2.4 Advanced routing Howto](#)⁶.

3.3.6.6. IP: GRE tunnel over IP

GRE est un protocole de tunnel qui a été originellement développé par CISCO, et qui peut réaliser plus de choses que le tunnel IP dans IP. Par exemple, on peut aussi transporter du trafic multi-diffusion et de l'IPv6 à travers un tunnel GRE. Un exemple complet de tunnel GRE pourra être consulté dans le [Linux 2.4 Advanced routing Howto](#)⁷.

3.3.6.7. IP: TCP Explicit Congestion Notification support

Fonctionnalité qui permet aux routeurs d'annoncer aux clients une congestion du réseau.

3.3.6.8. IP: TCP syncookie support

Prévient d'une attaque qui s'appelle le "SYN Flooding".

3.3.6.9. IP: Allow large windows (not recommended if <16 Mb of memory)

Permet de définir de plus gros tampons dans lesquels les données sont stockées avant d'être envoyées à l'hôte destinataire.

3.3.6.10. IP: Netfilter configuration

Permet de rentrer dans un nouveau menu pour la configuration du filtrage. Celui-ci permet l'ajout de fonctionnalités dont voici une liste des plus importantes :

Sous-option de Netfilter configuration

Connection tracking (required for masq/NAT)

Cette option permet de mettre en place le « Stateful ». Le « stateful » est la possibilité de garder en mémoire, dans une table d'état, les connexions en cours. Cela permet de différencier le trafic entre les mêmes machines, émettrice et réceptrice.

IP tables support (required for filtering/masq/NAT)

⁶ <http://www.linux-france.org/prj/inetdoc/i/net/guides/Advanced-routing-Howto/Advanced-routing-Howto.v130-5.html#ss5.2>

⁷ <http://www.linux-france.org/prj/inetdoc/i/net/guides/Advanced-routing-Howto/Advanced-routing-Howto.v130-5.html#ss5.3>

Cette option permet la mise en place de la structure générale pour le filtrage, le masquage ou la traduction d'adresse des paquets.

limit match support

Permet de limiter la fréquence à laquelle une règle est consultée.

Exemple 3.13. Limitation de consultation d'une règle

But : limiter les « pings » à 10 paquets par seconde.

```
$ iptables -A INPUT -p icmp -icmp-type 8 -m limit --limit 1/second -j ACCEPT
```

MAC address match support

Permet de baser le filtrage en considérant l'adresse source ethernet des trames entrantes.

Exemple 3.14. Filtrage de paquets en considérant l'adresse MAC

But : interdire les paquets provenant d'une adresse MAC particulière.

```
$ iptables -A INPUT -m mac --mac-source 00:A0:24:A0:A4:11 -j DROP
```

netfilter MARK match support

Permet de baser le filtrage sur la marque d'un paquet. Le marquage d'un paquet est réalisé grâce à la cible MARK (-j MARK).

Exemple 3.15. Filtrage de paquets par marquage de paquets

But : interdire les paquets à destination du serveur web local en utilisant le marquage de paquets.

```
$ iptables -A PREROUTING -t mangle -p tcp --dport 80 -j MARK --set-mark=2
$ iptables -A INPUT -m mark --mark 2 -j DROP
```

multiple port match support

Permet de spécifier un ensemble de ports sources ou destinations TCP ou UDP.

TOS match support

Permet de baser le filtrage sur la valeur du champ TOS du paquet.

Exemple 3.16. Exemple de filtrage basé sur la valeur du champ TOS

But : marquer les paquets qui ont le champ TOS Minimize-Delay activé. Ce marquage pourra ainsi être utilisé pour le routage de ces paquets.

```
$ iptables -t mangle -A PREROUTING -m tos --tos Minimize-Delay -j MARK --set-mark=1
```

tcpmss match support

Permet de baser le filtrage sur la valeur du champ MSS du paquet.

Connexion state match support

Permet de baser le filtrage sur l'état des connexions.

Exemple 3.17. Filtrage de paquets en considérant l'adresse MAC

But : interdire les nouvelles connexions entrantes vers la machine locale, mais autoriser les connexions depuis la machine locale.

```
$ iptables -P INPUT DROP
$ iptables -A INPUT -i eth0 -m state --state NEW,INVALID -j DROP
```

```
$ iptables -A INPUT -i eth0 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Unclean match support

Cette option permet de repérer des paquets qui semblent douteux ou inhabituels.

Exemple 3.18. Exemple d'utilisation de unclean

But : interdire les paquets entrant mal formés.

```
$ iptables -A INPUT -m unclean -j DROP
```

Owner match support

Permet de baser le filtrage sur l'identifiant du processus local ayant créé le paquet. Cette option ne peut être utilisée que dans la chaîne OUTPUT.

Packet filtering

Cette option permet le support du filtrage de paquets. La table gérant le filtrage se nomme "filter" et possède les chaînes par défaut INPUT, FORWARD et OUTPUT. La sous-option « REJECT target support » ajoutera la cible REJECT.

Full NAT

Cette option permet le support de la traduction d'adresse. Cette option permet la traduction d'adresse source (SNAT) et la traduction d'adresse de destination (DNAT).

Exemple 3.19. Exemple de traduction d'adresse source

But : masquer l'adresse source des paquets sortant par l'interface eth0 avec l'adresse 200.200.200.1 (adresse de l'interface eth0).

```
$ iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 200.200.200.1
```

Exemple 3.20. Exemple de traduction d'adresse de destination

But : rediriger les paquets à destination d'un serveur web vers le proxy de la machine 200.200.200.1 qui écoute le port 8080.

```
$ iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to 200.200.200.1:8080
```

Packet Mangling

Cette option permet de modifier certains éléments du paquet. La sous-option « TOS target support » permet de modifier le champ TOS du paquet. La sous-option « MARK target support » permet de marquer les paquets. Ces marques pourront être utilisées par la suite pour, par exemple, imposer un routage particulier.

Exemple 3.21. Configuration du champ TOS

But : positionner le champ TOS à Minimize-Delay pour les clients telnet.

```
$ iptables -A PREROUTING -t mangle -p tcp --dport telnet -j TOS --set-tos Minimize-Delay
```

LOG target support

Cette option permet l'enregistrement des paquets grâce au démon syslogd.

Exemple 3.22. Log de paquets web

But : Logguer tous les paquets à destination d'un site web sortant d'une machine.

```
$ iptables -A OUTPUT -p tcp --dport 80 -j LOG --log-level 7 --log-prefix "Paquets WEB :"
```

3.3.7. 802.1Q VLAN Support

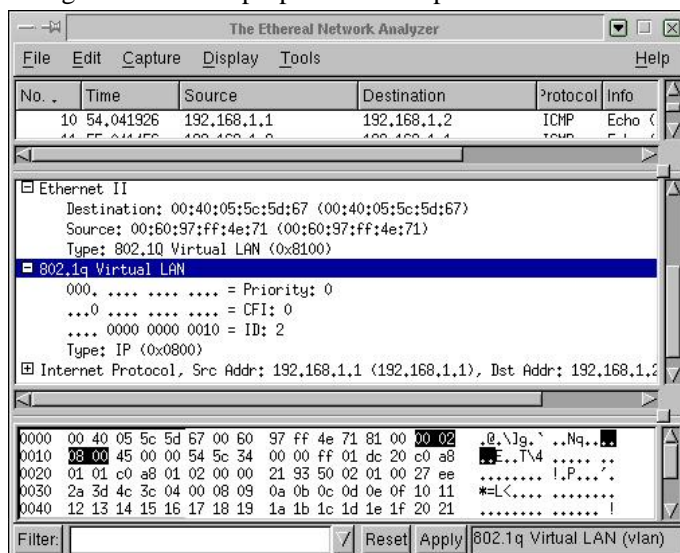
De nos jours, les LAN (Local Area Network) sont définis comme étant un domaine de diffusion unique. Les VLAN permettent de définir des réseaux locaux logiques sans se soucier de la localisation physique du matériel. Ces VLAN sont identifiés par un nombre. Suivant la configuration du commutateur, il est parfois nécessaire d'indiquer à quel VLAN appartient une trame qui est envoyée sur le réseau. C'est ici que le protocole 802.1Q intervient. Ce protocole va permettre le marquage des trames ethernet.

Exemple 3.23. Exemple de configuration du support VLAN

But : définir l'interface eth0 comme appartenant au VLAN numéro 2 machine.

```
LinuxBox# modprobe 8021q
LinuxBox# ifconfig eth0 0.0.0.0 up
LinuxBox# vconfig add eth0 2
LinuxBox# ifconfig eth0.2 192.168.1.1 up
```

La figure ci-dessous propose un exemple de trame ethernet avec le support du protocole 802.1Q.



3.3.8. The IPX Protocol

Ajoute le support du protocole IPX, principalement utilisé par les réseaux Novell. IPX est un protocole de couche 3 (couche réseau).

3.3.9. Appletalk DDP

Appletalk est le protocole de communication des ordinateurs Apple. Cette option permet à la machine Linux de pouvoir dialoguer avec les machines Apple. En utilisant le programme netatalk, Linux peut agir comme serveur d'impression et de fichiers pour Mac. Il pourra également accéder aux imprimantes Appletalk.

3.3.10. DECnet support

La société Digital a créé une architecture réseau complète qui porte le nom de DNA (Digital Network Architecture). Cette architecture s'appuie sur une pile de protocole qui englobe l'ensemble des couches du modèle OSI. Les produits qui implémentent l'architecture DNA sont appelés produits DECNet. Par abus de langage, on désigne parfois le terme DECNet pour identifier un réseau DNA. Pour plus d'informations sur ce sujet, consulter le site officiel du support [DECnet sous Linux](#)⁸.

3.3.11. 802.1d Ethernet Bridging

Avec cette option, votre boîte Linux pourra être assimilée à un pont Ethernet ; ce qui signifie que les différents segments Ethernet connectés apparaîtront comme un seul réseau Ethernet. Plusieurs ponts de ce type peuvent fonctionner ensemble pour créer un réseau de segments Ethernet encore plus grand en utilisant l'algorithme de Spanning Tree Protocol (IEEE 802.1). Comme le protocole 802.1 est standard, les ponts Linux fonctionneront correctement avec des équipements tiers.

Pour utiliser un pont Ethernet, vous aurez besoin des outils de configuration de pont. Voir `Documentation/networking/bridge.txt` pour plus d'information. Lire aussi le *BRIDGE-STP-HOWTO*⁹.

Notez que si votre machine fonctionne en pont, elle contient plusieurs interfaces Ethernet. Le noyau n'est pas capable de reconnaître plus d'une interface au démarrage sans assistance. Pour plus de détails, lire l'*Ethernet-HOWTO*¹⁰.

⁸ <http://linux-decnet.sourceforge.net/>

⁹ <http://www.linuxdoc.org/HOWTO/BRIDGE-STP-HOWTO/>

¹⁰ <http://www.linuxdoc.org/HOWTO/Ethernet-HOWTO.html>

Chapitre 4. Les outils réseaux du noyau Linux

Le code des outils de configuration réseau ne faisant pas partie du noyau est généralement appelé : *userspace code*.

4.1. Configuration des interfaces réseaux

A partir de la version 2.2, de nombreuses fonctionnalités sont apparues dans le protocole TCP/IP, notamment au niveau du routage. Pour pouvoir utiliser ces nouveautés, les outils classiques tels que **ifconfig** ou **route** ne suffisent plus. Il convient d'utiliser un nouvel outil, appelé **iproute2**.

La syntaxe générale pour l'outil **iproute2** est la suivante :

```
Usage: ip [ OPTIONS ] OBJET { COMMAND | help }
où OBJET := { link | addr | route | rule | neigh | tunnel |
             maddr | mroute | monitor }
OPTIONS := { -V[ersion] | -s[tatistics] | -r[esolve] |
             -f[amily] { inet | inet6 | dnet | link } | -o[neline] }
```

Les différents objets permettent de voir ou de configurer un élément du réseau.

4.1.1. ip link

L'objet **link** permet de visualiser l'état des périphériques réseaux et de les modifier. La syntaxe générale pour cette option est la suivante :

```
Usage: ip link set DEVICE { up | down | arp { on | off } |
                          dynamic { on | off } |
                          multicast { on | off } | txqueuelen PACKETS |
                          name NEWNAME |
                          address LLADDR | broadcast LLADDR |
                          mtu MTU }
ip link show [ DEVICE ]
```

Cette option ne s'intéresse qu'au niveau 2 du modèle OSI.

4.1.2. ip address

Cet objet permet d'attacher une ou plusieurs adresses IPv4 ou IPv6 à un périphérique réseau.

```
Usage: ip addr {add|del} IFADDR dev STRING
ip addr {show|flush} [ dev STRING ] [ scope SCOPE-ID ]
                    [ to PREFIX ] [ FLAG-LIST ] [ label PATTERN ]
IFADDR := PREFIX | ADDR peer PREFIX
         [ broadcast ADDR ] [ anycast ADDR ]
         [ label STRING ] [ scope SCOPE-ID ]
SCOPE-ID := [ host | link | global | NUMBER ]
FLAG-LIST := [ FLAG-LIST ] FLAG
FLAG := [ permanent | dynamic | secondary | primary |
         tentative | deprecated ]
```

La configuration de base d'une interface réseau ressemble à ceci :

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0 broadcast 192.168.0.255
route add -net 192.168.0.0 dev eth0
route add default gw 1.2.3.4
```

Les commandes équivalentes avec l'outil **iproute2** sont les suivantes :

```
ip addr add 192.168.0.1/24 dev eth0 broadcast 192.168.0.255
ip route add default dev eth0 via 1.2.3.4
```

4.1.3. ip rule

Le routage du trafic IP a été complètement revu avec le noyau 2.2. Avant cette version, la prise de décision ne se faisait qu'en consultant l'adresse de destination. Dans certaines circonstances, on peut souhaiter router les paquets IP en se basant sur d'autres champs : adresse source, champs TOS, etc...

Le routage est maintenant basé sur l'existence d'un ensemble de règles, qui dirigent le paquet vers des tables de routage. L'ensemble de ces règles est vu comme une base de données par le noyau, que l'on appelle *Routing Policy DataBase* (RPDB). Cette base de données de la politique de routage est en fait une liste linéaire de règles ordonnées par une valeur numérique de priorité. La gestion de ces règles se fait par l'intermédiaire de l'objet « rule », dont voici la syntaxe :

```
Usage: ip rule [ list | add | del ] SELECTOR ACTION
SELECTOR := [ from PREFIX ] [ to PREFIX ] [ tos TOS ] [ fwmark FWMARK ]
           [ dev STRING ] [ pref NUMBER ]
ACTION := [ table TABLE_ID ] [ nat ADDRESS ]
           [ prohibit | reject | unreachable ]
           [ realms [SRCREALM/]DSTREALM ]
TABLE_ID := [ local | main | default | NUMBER ]
```

Chaque règle est constituée d'un sélecteur et d'une action. Quand le noyau a besoin de prendre une décision sur le routage, la *Routing Policy DataBase* (RPDB) est scannée dans l'ordre des priorités croissantes. Pour chaque paquet, on compare le sélecteur de la règle et l'en-tête du paquet. Si il y a correspondance entre les deux, l'action est réalisée. En général, l'action consiste à se « brancher » sur une table de routage qui contient l'information utile. Si l'action ne parvient pas à déterminer une route, alors la règle suivante est examinée. La commande suivante permet de lister l'ensemble des règles définies dans la « RPDB » :

```
$ip rule ls
0: from all lookup local
32766: from all lookup main
32767: from all lookup default
```

Cette sortie mérite quelques explications. Les chiffres de la colonne de gauche indiquent la priorité de la règle. Ensuite, on a le sélecteur. Dans ce cas, toutes les règles seront appliquées à tous les paquets ("from all"). Enfin, on a l'action. Le mot-clé "lookup" indique d'aller regarder la table de routage dont le nom suit.

4.1.4. ip route

Une fois que le noyau a sélectionné la table à consulter, il recherche dans celle-ci les informations de routage proprement dites. Ces informations précisent le périphérique de sortie et éventuellement l'adresse de la prochaine passerelle. Par défaut, il y a trois tables de routage : local, main et default.

1. **Local** : cette table est une table un peu spéciale ayant la plus grande priorité. Elle contient les routes pour les adresses locales et les adresses de diffusion.
2. **main** : cette table est la table de routage normale, et ce sont les informations contenues dans celle-ci qui seront affichées par la commande **ip route ls**.
3. **default** : cette table est généralement vide et n'est consultée que si les règles précédentes n'ont pas sélectionné le paquet.

La syntaxe associée à l'objet route est la suivante :

```
Usage: ip route { list | flush } SELECTOR
       ip route get ADDRESS [ from ADDRESS iif STRING ]
                          [ oif STRING ] [ tos TOS ]
       ip route { add | del | change | append | replace | monitor } ROUTE
SELECTOR := [ root PREFIX ] [ match PREFIX ] [ exact PREFIX ]
           [ table TABLE_ID ] [ proto RTPROTO ]
           [ type TYPE ] [ scope SCOPE ]
ROUTE := NODE_SPEC [ INFO_SPEC ]
NODE_SPEC := [ TYPE ] PREFIX [ tos TOS ]
            [ table TABLE_ID ] [ proto RTPROTO ]
            [ scope SCOPE ] [ metric METRIC ]
INFO_SPEC := NH OPTIONS FLAGS [ nexthop NH ]...
NH := [ via ADDRESS ] [ dev STRING ] [ weight NUMBER ] NHFLAGS
OPTIONS := FLAGS [ mtu NUMBER ] [ advmss NUMBER ]
          [ rtt NUMBER ] [ rttvar NUMBER ]
          [ window NUMBER ] [ cwnd NUMBER ] [ ssthresh REALM ]
          [ realms REALM ]
TYPE := [ unicast | local | broadcast | multicast | throw |
         unreachable | prohibit | blackhole | nat ]
TABLE_ID := [ local | main | default | all | NUMBER ]
SCOPE := [ host | link | global | NUMBER ]
FLAGS := [ equalize ]
NHFLAGS := [ onlink | pervasive ]
RTPROTO := [ kernel | boot | static | NUMBER ]
```

4.2. Sélection et contrôle des services

4.2.1. Le démon inetd

Après la configuration des interfaces graphiques, il est important de décider des services que l'on désire offrir à Internet. Les services réseau sont assurés par des démons, qui sont des programmes qui «écoutent» un port et attendent des connexions de clients. Cette méthode oblige à lancer au démarrage tous les démons correspondant aux services que l'on souhaite proposer. Ceci induit un gachis des ressources système et de mémoire.

C'est pour cette raison que l'on emploie la plupart du temps un « super démon » qui écoute tous les ports des services proposés. Ce démon est le plus souvent appelé *inetd*. Celui-ci réceptionne les requêtes réseaux et lance les démons appropriés. Le fichier de configuration de *inetd* est `/etc/inetd.conf`. Ce fichier contient les noms des services dont *inetd* est à l'écoute et qu'il peut lancer.

Chaque service que l'on souhaite proposer correspondra à une ligne du fichier `inetd.conf`. Une ligne est composée des champs suivants :

```
service type protocole attente utilisateur serveur ligne-de-commandes
```

La signification des différentes lignes est la suivante :

- **service** : nom du service tel qu'il est déclaré dans le fichier `/etc/services`. Ce dernier fichier donne la correspondance entre le nom d'un service (par exemple ftp) et son numéro de port associé (21).
- **type** : spécifie le type de socket. La valeur `stream` sera donnée pour des services basés sur TCP, tandis que la valeur `dgram` sera fournie pour des services basés sur UDP.
- **protocole** : nom du protocole tel qu'il est déclaré dans le fichier `/etc/protocols`.
- **attente** : la valeur de ce champs sera toujours égale à `nowait` pour le protocole TCP. Dans le cas du protocole UDP, on peut avoir la valeur `wait` ou `nowait`. Dans le cas de `wait`, le serveur *inetd* attend que le serveur ait restitué le socket. Cela conduit à ce qu'un seul serveur ne soit lancé à la fois pour le port en question.
- **utilisateur** : nom de l'utilisateur sous lequel le processus sera lancé.
- **serveur** : chemin d'accès complet au programme serveur lancé par *inetd*.
- **ligne-de-commande** : arguments de la ligne de commande devant être transmis au programme serveur lorsqu'il est lancé. Cette liste commence toujours par le nom du programme.

4.2.2. Le contrôle des services

La conception du démon *inetd* impose soit l'accès soit le refus à un service quelle que soit la machine qui se connecte. Pour ne pas être aussi restrictif, il existe un programme, nommé **TCPd**, qui permet le contrôle d'accès. La mise en place de celui-ci permet d'accorder ou non l'accès à un service à la machine distante qui demande ce service. Pour mettre en place cette fonctionnalité, il convient de remplacer dans le fichier `/etc/inetd.conf` le chemin d'accès de chacun des démons de service réseau devant être sous contrôle d'accès, par le chemin de **tcpd**. Il faut donc modifier le sixième champ des lignes du fichier `/etc/inetd.conf`, comme le montre l'exemple suivant pour le démon *finger*.

Exemple 4.1. contrôle d'accès

```
finger stream tcp nowait nobody /usr/sbin/tcpd in.fingerd
```

Le contrôle de l'accès est géré au moyen de deux fichiers de configuration : `/etc/host.allow` et `/etc/host.deny`. Ces fichiers sont consultés dans cet ordre. Si une correspondance démon-client est trouvée dans le fichier `/etc/host.allow`, alors l'accès est autorisé et le démon est lancé. Sinon, le fichier `/etc/host.deny` est scruté. Si une correspondance démon-client est trouvé, l'accès est refusé. Enfin, si rien n'est trouvé, la requête est acceptée. Les entrées des deux fichiers de configuration doivent respecter le format suivant :

```
liste_de_démons : liste_de_clients
```

explication des champs `liste_de_démons` et `liste_de_clients`

`liste_de_démon`

liste de noms valides de services listés dans le fichier `/etc/services`, ou encore le mot clé `ALL` pour désigner tous les services.

`liste_de_clients`

liste de noms de machines ou d'adresses IP, ou les mots clé `ALL`, `LOCAL`, `UNKNOWN`. `ALL` indique n'importe quelle machine, `LOCAL` désigne des machines dont le nom ne contient pas de point et `UNKNOWN` fait référence à des machines dont la recherche de nom ou d'adresse a échoué.

Pour plus de précisions, consulter les pages de manuels `tcpd` (8) et `hosts_access` (5).

4.3. Configuration du filtrage

4.3.1. Introduction

La sécurité informatique est un terme général qui cache de nombreuses aspects, tels que la sécurité physique de la machine, le contrôle d'accès aux fichiers, etc... L'un des aspects de la sécurité concerne la sécurité des réseaux. Avec la démocratisation d'Internet, les tentatives d'intrusions se développent. Afin de limiter le nombre de ces attaques, le mieux est encore de filtrer dès l'entrée du réseau tout ce qui n'est pas sensé y être. Le système qui permet la mise en place de ce filtrage s'appelle un « Firewall » ou un « pare-feu » en français.

Un firewall peut se définir comme étant un dispositif de protection (matériel et/ou logiciel) constituant un filtre entre un ordinateur ou un réseau local et un réseau non sûr (Internet ou un autre réseau local par exemple). On distingue deux grandes familles de firewall :

- Les firewalls par filtrage de paquets : ces éléments fonctionnent au niveau de la couche réseau (couche 3) du modèle OSI. Le filtrage s'effectue en fonction des informations contenues dans les en-têtes des paquets (adresses source et destination, ports source et destination). Ce type de filtrage ne s'intéresse pas au contenu des paquets.
- Les firewalls par proxy de services : ce type de filtrage permet de contrôler les couches supérieures. Dans ce cas, l'information contenue dans le paquet peut être prise en compte. Les demandes de connexions sont dirigées vers un programme spécial appelé mandataire ou proxy de service. C'est ce dernier qui établira la connexion vers le service extérieur demandé.

Le filtrage de paquets consiste à regarder l'en-tête d'un paquet qui traverse une machine et à décider de l'avenir de ce paquet. Celui-ci peut être rejeté, accepté ou modifié suivant des règles plus ou moins complexes. Dans de nombreux cas, on utilisera le filtrage pour contrôler et/ou sécuriser un réseau interne du monde extérieur, par exemple Internet. Dans la littérature, on retrouvera souvent le terme de Firewall pour désigner une machine qui réalise ces fonctions de filtrage.

4.3.2. Architecture d'un firewall

4.3.3. Netfilter et iptables

Pour pouvoir bénéficier du filtrage sous Linux 2.4, il faut intégrer la fonctionnalité appelée Netfilter lors de la compilation. Cette fonctionnalité est une structure générale qui permet à d'autres éléments de se « brancher » dessus. Pour pouvoir indiquer les différentes règles au noyau, on dispose de l'utilitaire appelé **iptables**.

iptables utilise le concept de tables de règles, chaque table correspondant à une fonctionnalité de traitement du paquet. La table `filter` correspond au filtrage des paquets, la table `nat` concerne la traduction d'adresse et la table `mangle` permet la manipulation des paquets.

4.3.4. Utilisation de l'outil iptables

Dans un premier temps, iptables servira à la gestion des chaînes. Une chaîne peut être assimilée à une politique de sécurité associée à un flux de données. Par exemple, on peut définir une chaîne INTERNET pour désigner tous les flux venant de l'extérieur de votre réseau local. Trois chaînes par défaut existent, à savoir INPUT, FORWARD et OUTPUT. Si le nombre de règles est limité, on peut se contenter de celles-ci, mais si les règles deviennent conséquentes, il est préférable, pour faciliter la gestion, de créer de nouvelles chaînes. Les commandes de iptables associées à la gestion des chaînes sont les suivantes :

1. -N : création d'une nouvelle chaîne (iptables -N INTERNET)
2. -X : suppression d'une chaîne vide (iptables -X INTERNET)
3. -P : Mise en place de la règle par défaut pour une chaîne existante (iptables -P INPUT DROP). Seules les chaînes INPUT, FORWARD et OUTPUT peuvent avoir une règle par défaut et les seules cibles disponibles sont ACCEPT et DROP.
4. -L : lister les règles d'une chaîne (iptables -L INTERNET)
5. -F : effacer les règles d'une chaîne (iptables -F INTERNET)

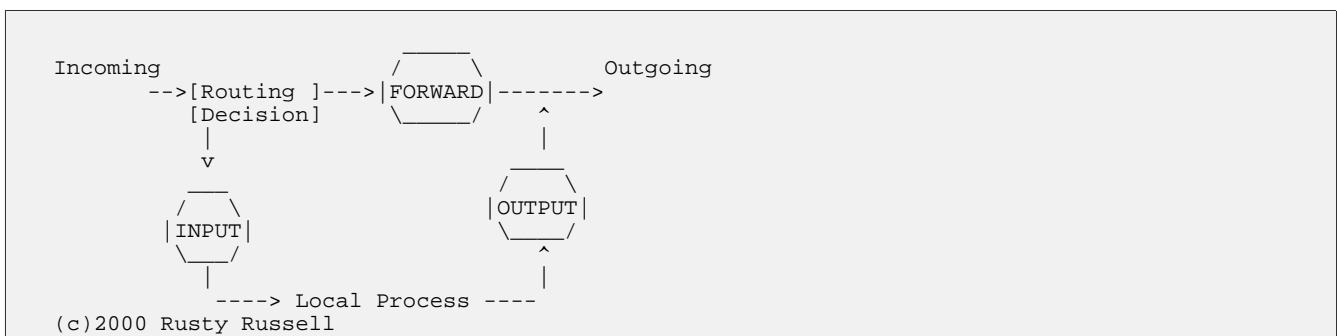
Dans un deuxième temps, il convient de construire les règles à l'intérieur des différentes chaînes. L'ajout d'une règle s'effectue avec l'option -A de iptables, tandis que l'effacement d'une règle se réalise avec l'option -D. Les principales spécifications sur lesquelles les règles peuvent s'appuyer sont les suivantes :

- -s : spécifie l'adresse IP source
- -d : spécifie l'adresse IP de destination
- -p : spécifie le protocole. Le protocole peut être tcp, udp ou icmp
- -i : spécifie le nom de l'interface physique à travers laquelle les paquets entrent
- -o : spécifie le nom de l'interface physique à travers laquelle les paquets sortent

Ces spécifications sont les plus générales, mais il en existe bien d'autres qui sont parfaitement listées dans la page de manuel de iptables.

4.3.5. Le filtrage avec iptables

Avec iptables, les différentes règles de filtrage sont organisées et regroupées dans des chaînes. Par défaut, il y a trois chaînes appelées INPUT, OUTPUT et FORWARD. L'arrangement de ces chaînes est proposé sur le schéma suivant :



Les différentes chaînes sont consultées suivant la procédure suivante :

1. Quand un paquet arrive, le noyau décide de la destination de ce paquet : c'est la phase de routage.
2. Si le paquet est destiné à la machine, le paquet descend dans le diagramme et la chaîne INPUT est appliquée. Si le paquet passe cette chaîne, celui-ci sera transmis à l'un des processus locaux.
3. Si le routage décide que le paquet est destiné à un autre réseau, alors c'est la chaîne FORWARD qui est appliquée.

4. Enfin, les paquets envoyés par un processus local seront examinés par la chaîne OUTPUT. Si le paquet est accepté, celui-ci sera envoyé quelle que soit son interface de sortie.

Une chaîne est composée d'une liste de règles. Une règle décide de l'avenir d'un paquet en fonction de son en-tête. Les règles d'une chaîne sont examinées les unes après les autres jusqu'à ce qu'une correspondance soit trouvée. Finalement, si aucune correspondance n'est trouvée, la règle par défaut, policy, est appliquée. On associe à chaque règle une action à réaliser qui décide de l'avenir du paquet. Les fonctions principales sont les suivantes :

- ACCEPT : cette cible permet d'accepter les paquets.
- DROP : cette cible permet de refuser les paquets sans avertir le demandeur que sa demande de connexion a été refusée.
- REJECT : cette cible permet de refuser les paquets, mais en avertissant le demandeur que sa demande de connexion a été refusée en lui envoyant un paquet RESET (RST).

4.3.6. Stateful, le suivi des connexions

L'une des grandes nouveautés de la partie réseau du noyau 2.4 est la possibilité du suivi des connexions. Ceci fait référence à la capacité du noyau à maintenir une table de suivi des connexions en se basant, par exemple, sur le couple adresses (source et destination) et numéros de ports (source et destination), sur les types de protocoles ou l'état de la connexion. Les firewalls disposant de cette fonctionnalité sont appelés firewall stateful. Dans ce cas, les paquets sont inspectés dans le contexte d'une session. Par exemple, un paquet TCP avec le bit ACK activé sera rejeté si aucun paquet SYN correspondant n'a été perçu.

Le suivi des connexions se base sur trois états de ces connexions :

- NEW : correspond à la requête TCP SYN initiale ou au premier paquet UDP.
- ESTABLISHED : si une entrée de la table de suivi des connexions correspond, alors le paquet appartient à une connexion de type ESTABLISHED. Cela se réfère au paquet TCP ACK après qu'une connexion ait été initiée, à un datagramme UDP échangé entre deux hôtes et numéros de ports identiques ou à un message ICMP echo-reply envoyé en réponse à un message echo-request.
- RELATED : se réfère aux messages d'erreurs ICMP correspond à une connexion tcp ou udp déjà présente dans la table de suivi.

D'un point de vue pratique, le module de suivi des connexions sera activé grâce à l'option `-m state` de **iptables**. L'option `--state` permet de spécifier l'état de la connexion à considérer.

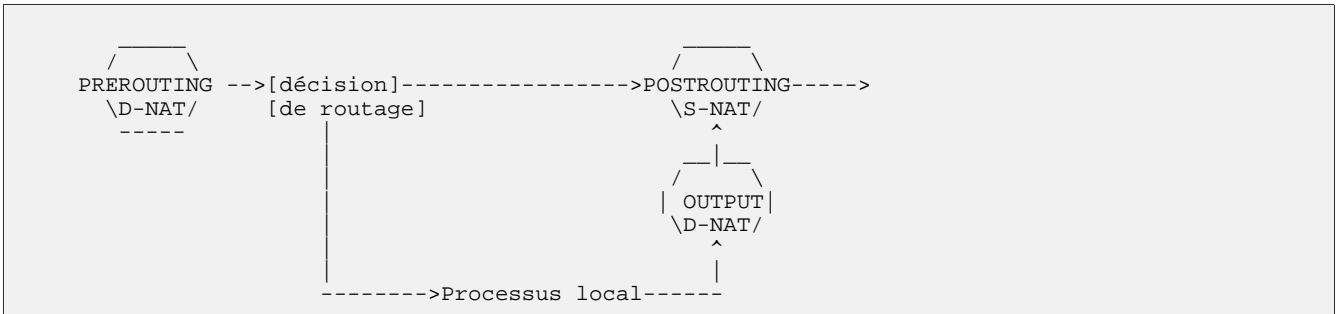
Exemple 4.2. Autorisation d'une connexion ssh vers l'extérieur

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -p tcp -d 0/0 --dport 22 -m state --state NEW -j ACCEPT
```

4.3.7. La traduction d'adresse (NAT)

La traduction d'adresse est une technique qui permet de remplacer une adresse source ou destination par une autre. La traduction d'adresse du noyau 2.4 supporte le source NAT (SNAT) et la destination NAT (DNAT). La table nat permet la modification des adresses source et destination grâce à trois chaînes par défaut :

- PREROUTING : permet la modification de l'adresse de destination (DNAT) avant que le paquet ne passe par les fonctions de routage.
- OUTPUT :
- POSTROUTING : permet la modification de l'adresse source (SNAT) après que le paquet ait passé par les fonctions de routage.



Intéressons-nous dans un premier temps au Source NAT. Il existe en deux formes distinctes au sein du noyau 2.4 : SNAT et MASQUERADE. SNAT est la forme standard de la traduction d'adresse source, tandis que la deuxième est plus spécialisée au cas d'adresses IP assignées dynamiquement. La distinction entre les deux formes est subtile. Avec SNAT, la connexion est maintenue pendant un certain temps d'attente lors d'un dysfonctionnement. Si cette connexion est rétablie suffisamment rapidement, les programmes réseaux ne seront pas affectés et le trafic TCP interrompu sera retransmis, dans la mesure où l'adresse IP n'a pas changé. Avec la forme MASQUERADE, il n'y a pas de temps d'attente quand la connexion est rompue et les informations concernant le NAT sont effacées. Ceci permet d'utiliser immédiatement la nouvelle adresse IP qui peut être attribuée lors d'une reconnexion à un fournisseur d'accès, par exemple.

Exemple 4.3. SNAT standard

```
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 192.192.192.192
```

Exemple 4.4. MASQUERADE

```
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```