22 août 2008

Le noyau et le réseau : Comment repousser les limites de la connectivité

Catégorie : Administration réseau Tags : lmhs

~~Retrouvez cet article dans :~~ Linux Magazine Hors série 17

Parmi les domaines dans lesquels excelle le noyau Linux, il y a incontestablement le réseau. Qu'on regarde les matériels supportés via les drivers présents dans le noyau, les protocoles disponibles ou les nombreuses fonctionnalités, on ne peut que se rendre à une évidence : Linux un des systèmes d'exploitation le plus interopérable du marché au niveau réseau.

Organisation des supports réseau

Tout au long de cet article, je ferai référence au menu de configuration d'un noyau de la série 2.6, le ~~2.6.0-test5~~. Comme à l'accoutumée, l'arbre des sources se trouvera dans ~~/usr/src/linux-2.6.0-test5~~ avec un petit lien symbolique linux pointant vers ce répertoire. On n'aura évidemment pas oublié de vérifier la signature de l'archive avant de commencer à travailler.

```
cbr@elendil:/usr/src$ ls -la
total 40520
drwxr-xr-x    3 root     src          4096 2003-09-28 18:09 ./
drwxrwxrwt    6 root     root         4096 2003-09-28 18:05 ../
```

```
lrwxrwxrwx    1 root     src           17 2003-09-28 18:05 linux -> linux-2.6.0-test5/
drwxr-xr-x   18 root     src         4096 2003-09-21 19:54 linux-2.6.0-test5/
-rw-r--r--    1 root     src     41430080 2003-09-08 22:33 linux-2.6.0-test5.tar.gz
-rw-r--r--    1 root     src          248 2003-09-08 22:33 linux-2.6.0-test5.tar.gz.sign
```

Nous nous plaçons dans le répertoire ~~linux-2.6.0-test5~~ et nous nous préparons à configurer notre noyau en vue de sa compilation en lançant un ~~make menuconfig~~. En cas de doute, n'hésitez pas à consulter l'aide et les fichiers de documentation fournis dans ~~/usr/src/linux/Documentation/networking/~~.
Nous commencerons par activer le support des parties de codes expérimentales ou incomplètes (Code maturity level options, Prompt for development and/or incomplete code/drivers). De nombreuses fonctionnalités réseau dépendent en effet de cette option de configuration (~~CONFIG_EXPERIMENTAL~~), bien qu'elles soient suffisamment stables pour une utilisation sur des plates-formes de production. Selon ses goûts, les supports pourront être compilés à même le noyau (~~builtin~~) ou en modules. Nous ne discuterons pas ici des avantages et inconvénients de l'une ou l'autre de ces deux approches, mais rappellerons que l'utilisation des modules pour tester des fonctionnalités peut être une bonne idée.
Enfin, nous activerons le support Sysctl (configuration de paramètres noyau par modification de valeurs dans ~~/proc/sys/~~, cf General setup, Sysctl support) puisque de très nombreux paramètres réseau sont ajustables de cette manière.
Dans les noyaux 2.6, l'ensemble des options concernant le réseau se trouve sous la section Networking support (alors que les drivers et les fonctionnalités étaient séparées dans les noyau 2.4). C'est donc à cette section que nous allons nous intéresser tout au long de cet article. D'autres fonctionnalités se trouvent dispersées dans d'autres sections, j'en parlerai en fin d'article. Nous activons donc le support réseau (CONFIG_NET) et entrons dans le vif du sujet, comme montré dans la figure 1.

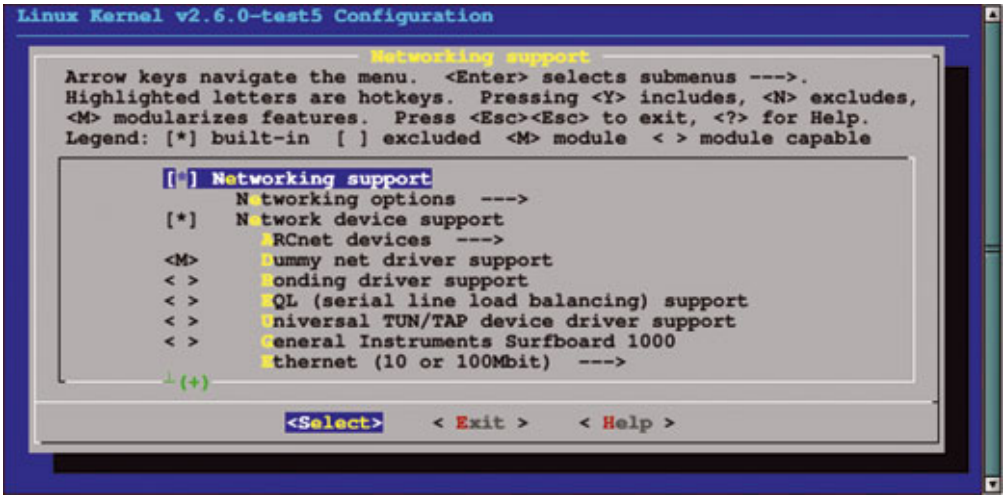

Fig. 1 : Configuration du réseau.

Cet écran de configuration comprend deux sections majeures :

- Networking options, qui regroupe les fonctionnalités réseau ;
- Networking device support, en dessous de laquelle on trouvera les pilotes réseau.

L'article n'a pas pour objet la description exhaustive de toutes les options et sous-options de la section réseau. Pour chacune d'entre elles, je vous invite à lire la l'aide accessible via le bouton <Help> en bas à droite de l'écran de configuration. Cependant, nous allons nous attarder sur certaines qui me semblent particulièrement intéressantes.

Les pilotes réseau

Nous trouvons ici tous les types d'interface réseau que peut supporter Linux, des désormais omniprésentes cartes Ethernet (10/100Mbps, 1Gbps ou 10Gbps) aux interfaces WAN en passant par les cartes Token Ring et les liens PPP. Nous allons nous focaliser sur certains d'entre eux.

Bonding driver support

Le pilote ~~bonding~~ permet de réaliser de l'agrégat de plusieurs interfaces Ethernet (EtherChannel dans la littérature Cisco). Ainsi, si vous disposez de trois interfaces Ethernet 100Mbps, vous pouvez les agréger en une seule interface logique disposant d'une bande passante de 300Mbps.
Pour configurer cela, il faut d'abord charger le module :

```
cbr@elendil:~# modprobe bonding
```

Ce driver peut recevoir des paramètres qui sont décrits dans le fichier ~~/usr/src/linux/Documentation/networking/bonding.txt~~. On pourra aussi aller consulter le site dédié à cette fonctionnalité [1]. Une fois le module chargé, nous pouvons commencer à agréger nos interfaces 100Mbps. Cette opération est réalisée par l'outil ~~ifenslave~~, dont le source est distribué avec le noyau (~~/usr/src/linux/Documentation/networking/ifenslave.c~~).

```
cbr@elendil:~# ifconfig bond0 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
cbr@elendil:~# ifconfig eth0 up
cbr@elendil:~# ifenslave bond0 eth0
cbr@elendil:~# ifconfig eth1 up
cbr@elendil:~# ifenslave bond0 eth1
cbr@elendil:~# ifconfig eth2 up
cbr@elendil:~# ifenslave bond0 eth2
```

Nous disposons maintenant d'une interface nommée ~~bond0~~ disposant d'une bande passante de 300Mbps. La distribution des trames sur les interfaces se fait par ~~round-robbin~~ en utilisant comme adresse MAC l'adresse de la première interface ajoutée à l'agrégat (eth0 dans cet exemple). Bien évidemment, pour fonctionner

au mieux, cette fonctionnalité suppose la présence à l'autre bout d'un équipement convenablement configuré. Sur un Cisco 2950 [2], nous aurons besoin d'une configuration de ce genre :

```
2950# configure terminal
2950(config)# interface port-channel 1
2950(config)# interface range fastethernet0/1 -3
2950(config-if)# channel-group 1 mode on
2950(config-if)# end
```

Enfin, nous pouvons consulter les information relatives à notre agrégat dans le répertoire ~~/proc/net/bond0~~ :

```
root@elendil:/proc/net/bond0# ls
info
root@elendil:/proc/net/bond0# cat info
Bonding Mode: load balancing
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: eth2
MII Status: up
Link Failure Count: 0

Slave Interface: eth1
MII Status: up
Link Failure Count: 0

Slave Interface: eth0
MII Status: up
Link Failure Count: 0
```

Notons que ce driver supporte la tolérance aux pannes à l'aide d'un monitoring MII (paramètre ~~miimon~~) et le failover sur un ou plusieurs ~~switches~~ (paramètre ~~mode=1~~), ainsi que le protocole 802.3ad (~~Dynamic link aggregation~~). Je vous renvoie à la documentation pour de plus amples détails.

Universal TUN/TAP device driver

Ce driver permet la création d'une interface logique de type point à point (~~tun0~~ ou ~~tap0~~). Le trafic qui transite par cette interface se retrouve en entrée ou sortie du fichier ~~/dev/net/tun~~ sous forme de paquets IP (~~TUN~~) ou des trames Ethernet (~~TAP~~). Ainsi, une application peut injecter du trafic dans le noyau pour le voir apparaître à travers l'interface associée ou récupérer les paquets émis par cette interface, comme illustré en figure 2.
D'un côté, l'application servant de base à l'interface doit ouvrir le périphérique :

```
#include <net/if.h>
#include <linux/if_tun.h>
```

```
int tun_alloc(char *dev)
{
        struct ifreq ifr;
        int fd, err;

        if ( (fd = open("/dev/net/tun",O_RDWR)) < 0)
                return tun_alloc_old(dev);

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

        ifr.ifr_flags = IFF_TUN;
        if ( *dev )
                strncpy(ifr.ifr_name, "dev", IFNAMSIZ);

        if( (err=ioctl(fd, TUNSETIFF, (void *)&ifr) < 0){
                close(fd);
                return err;
        }
        strcpy(dev, ifr.ifr_name);
        return fd;
}
```

Ensuite, elle écrira ses trames, au format désiré selon la valeur du paramètre ~~ifr.ifr_flags (IFF_TUN, IFF_TAP ou IFF_NO_PI)~~. De l'autre côté, nous obtenons une interface de type point à point (~~tap0~~) ou de type Ethernet (~~tun0~~), que nous pouvons configurer comme une interface classique :

```
cbr@elendil:~# ifconfig tun0 192.168.1.1 netmask 255.255.255. broadcast 192.168.1.255
```

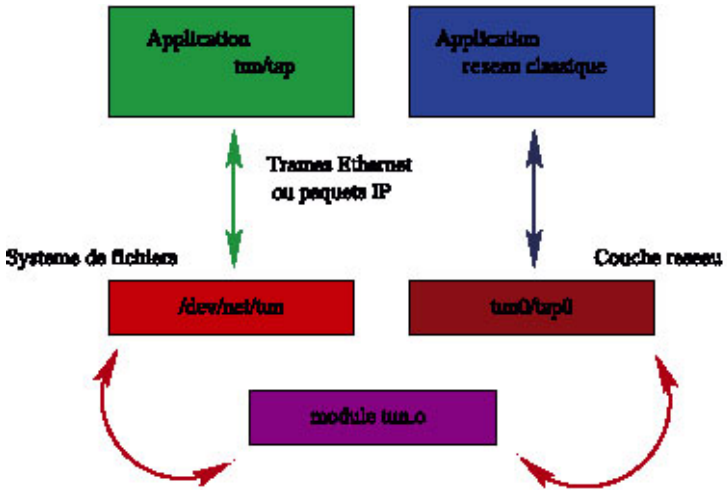

Fig. 2 : Driver TUN/TAP.

Ce type de configuration est utilisé pour des simulations d'environnement réseau, à des fins de développement par exemple, ou pour des applications visant à mettre en place des tunnels non standards comme ~~tunproxy~~ [3], ~~vtun~~ [4] ou encore OpenVPN [5]. Notons qu'un driver similaire existe pour Solaris, FreeBSD, et même pour les environnements Win32 (distribué avec le portage pour Win32

de OpenVPN). L'avantage d'un tunnel construit avec TUN/TAP est qu'il vous permet d'envoyer des trames de niveau 2, et donc de répartir un LAN, plutôt que de lier des réseaux IP différents.
Pour plus de détails, voir /usr/src/linux/Documentation/networking/tuntap.txt ou le site officiel dédié au driver [6].

PPP (point-to-point protocol)

Le protocole PPP est bien évidemment supporté par Linux, ainsi que les options courantes qui l'accompagnent comme la compression (de type Deflate ou BSD) et la gestion des liens synchrones (HDLC) et asynchrones. Le support du PPPoE est intégré au noyau, ce qui permet, moyennant l'utilisation d'un pppd [7] adéquat (v2.4.2 en CVS), de supporter ce type d'encapsulation au niveau noyau plutôt qu'en espace utilisateur avec un client comme le RoaringPenguin [8] et ainsi gagner en performances.
L'option PPP filtering permet quant à elle de configurer des filtres sur les trames PPP directement au niveau de pppd (cf la page man de pppd), permettant ainsi de choisir les trames pouvant circuler sur le lien, tant au niveau client qu'au niveau serveur. Enfin, l'option PPP multilink permet d'agréger des liens PPP physiques (canaux B en RNIS par exemple) ou logiques en une seule interface. Ainsi, si vous montez un canal PPP sur IP et que vous disposez de plusieurs routes possibles pour atteindre l'autre partie, vous pouvez lancer un pppd avec l'option multilink par route et obtenir un agrégat de bande passante supérieure. Comme pour le driver bonding vu précédemment, la distribution des trames se fait par round robbin entre les instances de pppd participant au lien agrégé. Ce mode de gestion a son importance. Si vous disposez en effet de deux liens de latence très différente, votre lien PPP global s'en trouvera fortement affecté, le lien lent ralentissant l'ensemble de l'agrégat.

Traffic Shaper

Le module shaper vous permet d'obtenir un outil de limitation de bande passante simple, mais limité. À vous de voir s'il pourra répondre à vos besoins, mais en ce qui me concerne, je vous recommande plutôt l'utilisation des options de qualité de service disponibles sous QoS and/or fair queueing. Pour s'en servir, on utilise l'outil shapecfg (disponible en paquetage dans la plupart des distributions) :

```
cbr@elendil:~# modprobe shaper
cbr@elendil:~# shapecfg attach shaper0 eth0
cbr@elendil:~# shapecfg speed shaper0 64000
cbr@elendil:~# ifconfig shaper0 192.168.1.1 netmask 255.255.255. broadcast 192.168.1.255
```

Ensuite, il vous suffira de router le trafic à limiter à travers shaper0. Simple, rustique, mais moyennement efficace. Parce qu'il utilise le mécanisme de routage, il ne peut limiter que le trafic sortant, et non le trafic entrant. Enfin, il n'y a pas de mécanisme de partage ou d'emprunt de bande passante comme c'est

le cas avec des systèmes plus élaborés comme CBQ ou HTB, que je vous conseille. Pour plus de détails, voir /usr/src/linux/Documentation/networking/shaper.txt.

Les fonctionnalités réseau

Maintenant que nous avons pu voir les interfaces, physiques ou logiques, que pouvait supporter notre noyau, voyons ce que nous pouvons en faire, en nous focalisant sur le monde IP. Nous entrons donc dans la section Networking options. Là encore, je ne détaillerai pas toutes les options et vous renvoie donc aux descriptions accessibles via l'aide.
Options de base
Pour obtenir un noyau capable de réaliser un nombre suffisant d'opération, nous allons activer certaines options :

- Packet socket : sockets de type pf_packet utilisées pour accéder directement aux périphériques réseau, dont se servent par exemple les bibliothèques de capture libpcap [9] ou de capture/injection libnet [10] ;
- Packet socket: mmapped IO : optimisation ;
- Netlink device emulation : option de compatibilité destinée à disparaître permettant aux applications utilisant /dev/tap0 (ancien driver EtherTap) ou /dev/route (démon de routage dynamique) de fonctionner ;
- Unix domain sockets : les fameuses sockets de type Unix ;
- TCP/IP networking : on veut faire de l'IP, donc on active le support adéquat ;
- IP Multicasting : si vous voulez faire du multicast IP ;
- Network packet filtering : support de Netfilter[11].

Avec ceci, vous avez de quoi faire de l'IP dans de bonnes conditions. Nous allons maintenant explorer des fonctionnalités plus avancées pour transformer votre Linux en bête de réseau.

IP advanced router

Votre Linux est déjà capable de router des paquets IP. Il suffit pour cela d'activer l'IP forwarding via le Sysctl (et de vérifier que vos tables de routage et votre filtrage IP sont cohérents) :

```
cbr@elendil:~# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Vous disposez alors d'un routeur simple, c'est-à-dire n'utilisant que l'adresse de destination de paquets pour choisir une route. En activant l'option "Advanced Router", nous allons pouvoir aller beaucoup plus loin et contrôler notre routage de manière très fine. Les options suivantes sont à notre disposition :

- Policy routing : cette option nous permet d'introduire de nouveaux critères de routage, comme le routage sur l'adresse source des paquets. En outre, nous voyons apparaître deux nouvelles options : use netfilter MARK value as

routing key et fast network address translation. La première est particulièrement intéressante. En marquant les paquets à l'aide de la cible ~~MARK~~ de Netfilter[11], nous sommes capable de modifier le routage de nos paquets. En clair, cela veut dire que tout élément reconnaissable par Netfilter peut nous servir de critère de routage. La seconde permet de mettre en place du NAT au niveau de la couche de routage. Cette traduction d'adresses est nettement moins souple que celle fournie dans Netfilter, mais se révèle particulièrement rapide. Ceux qui avaient l'habitude de faire du NAT (pas seulement du masquerading) avec les noyaux 2.2 ont déjà utilisé ce mécanisme.

- ~~Equal cost multipath~~ : cette option vous permet d'avoir à un même instant deux routes de même poids vers la même destination. Le choix de la route est fait de manière non déterministe (la décision dépend entre autres de l'état du cache de routage) parmi toutes les routes disponibles. C'est la base des configurations de partage de charge s'appuyant sur le routage.

- ~~Use TOS value as routing key~~ : le critère de routage est le champ TOS des paquets IP. Intéressant. On pourra noter que la valeur du champ TOS peut être fixée par Netfilter avec la cible TOS de la même manière qu'on marque les paquets. On peut ainsi transformer le champ TOS en marque exportable sur le réseau (ce qui n'est pas le cas des marques Netfilter).
- ~~Verbose route monitoring~~ : cette option permet d'obtenir des messages supplémentaires dans les journaux d'activité (via ~~klogd~~) sur les problèmes de routage. Extrêmement utile pour déboguer un routage dynamique qui ne fonctionne pas.

L'utilisation d'un système Linux en routeur avancé nécessite l'installation du paquetage iproute2 [12]. Je vous recommande chaudement la lecture (et la relecture) de la mine d'information qu'est le ~~Linux Advanced Routing and Traffic Control HOWTO [13]~~, plus communément appelé LARTC.
N'oubliez pas, sauf configuration particulière (routage asymétrique), d'activer le Reverse Path Filtering sur vos interfaces, c'est-à-dire le contrôle, par rapport à la table de routage, de l'interface d'arrivée de chaque paquet. Toute incohérence entraînera la destruction du paquet. C'est la base de l'~~antispoofing~~. Et comme vous voulez loguer ces échecs, activez la journalisation des "martiens" :

```
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
        echo 1 > $f
done
for m in /proc/sys/net/ipv4/conf/*/log_martians; do
        echo 1 > $m
done
```

Baladez-vous un peu dans ~~/proc/sys/net~~. Vous verrez qu'on peut y agir sur de nombreuses fonctionnalités (ainsi que leurs paramètres éventuels) de votre noyau.

IP kernel level autoconfiguration

Cette fonctionnalité est intéressante pour les stations devant monter leur système de fichiers racine par le réseau (stations diskless en particulier). Le noyau de ces stations pourra ainsi obtenir sa configuration IP directement du réseau, en DHCP, BOOTP ou RARP, sans avoir à passer par un quelconque fichier de configuration ou autre client applicatif. Un must have pour "booter" sur le réseau.

IP tunneling et GRE tunnels over IP

Ces deux options permettent respectivement l'utilisation de tunnel IP sur IP (IPIP) et GRE (Generic Routing Encapsulation) sur IP. Ce sont des tunnels simples, sans chiffrement ni authentification forte, mais permettant de résoudre des problématiques comme les migrations de sites. Pour monter un tunnel, vous avez besoin du paquetage ~~iproute2~~ [12].

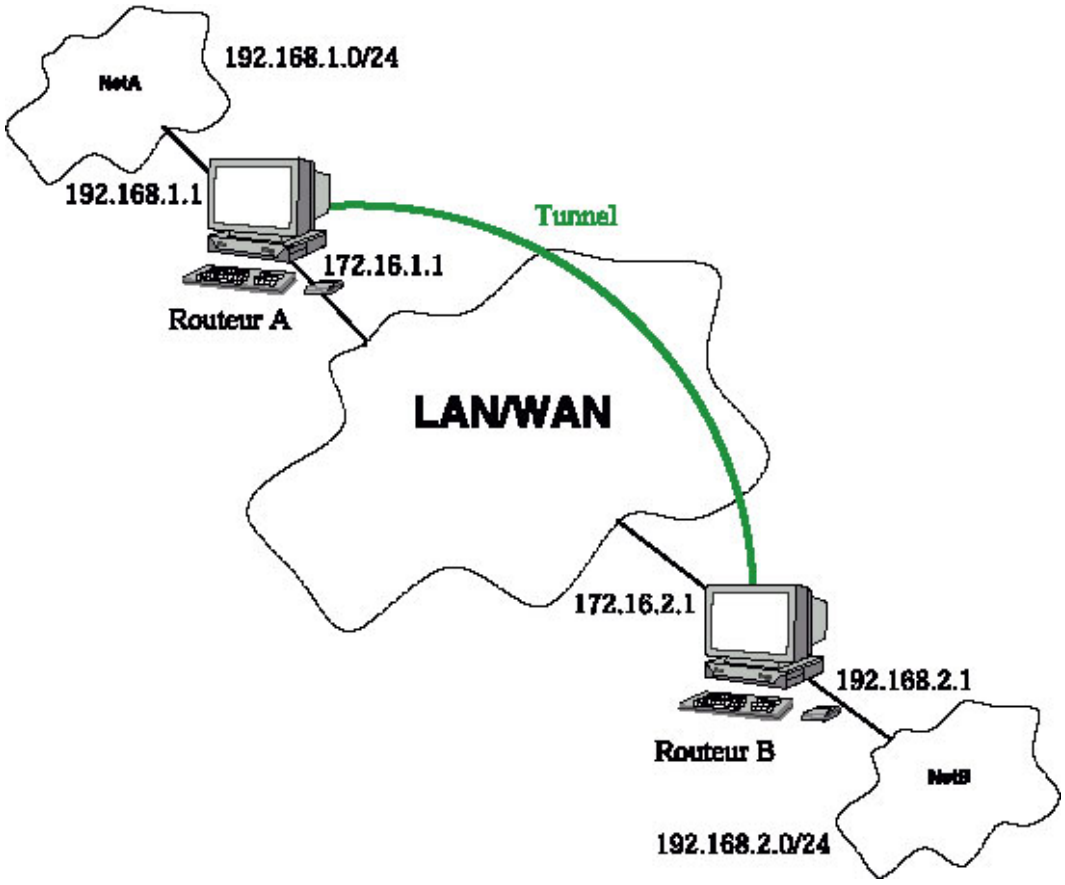

Fig. 3 : Mise en place d'un tunnel.

Pour monter un tunnel IPIP, nous devons avant toute autre chose charger les

modules concernés :

```
cbr@elendil:~# modprobe ipip
cbr@elendil:~# modprobe new_tunnel
```

À ce stade-là, nous disposons d'une interface tunl0 (et une seule) que nous pouvons configurer. Supposons que nous disposons de la configuration décrite en figure 3. Nous voulons monter un tunnel reliant les réseaux NetA et NetB via les routeurs A et B. Sur le routeur A, nous configurons le tunnel ainsi :

```
cbr@routerA:~# ifconfig tunl0 10.0.0.1 pointopoint 172.16.2.1
cbr@routerA:~# route add -net 192.168.2.0/24 dev tunl0
```

Sur le routeur B, nous aurons :

```
cbr@routerB:~# ifconfig tunl0 10.0.0.2 pointopoint 172.16.1.1
cbr@routerB:~# route add -net 192.168.1.0/24 dev tunl0
```

Et c'est bon, on peut y aller. On doit évidemment activer le routage et mettre en place un filtrage cohérent avec l'utilisation du tunnel. Mais le mode IPIP est limité. En effet, comme vous avez pu le constater, nous n'avons qu'un tunnel sous la main. Si nous voulons en mettre plusieurs en place, il faudra passer par un applicatif, s'appuyant par exemple sur le driver TUN/TAP. En outre, un tunnel IPIP, comme son nom l'indique, ne peut faire passer que de l'IP, seulement sur IP (M. de la Palisse ne m'aurait certainement pas contredit sur ce point) et uniquement à destination d'un autre système Linux, ce mode d'encapsulation lui étant (pour autant que je sache) spécifique. Malgré ces limitations, il n'en reste pas moins un mode de tunneling simple et efficace.
Pour des besoins d'interopérabilité ou pour encapsuler de l'IPv6, on s'orientera plutôt vers GRE, un standard initialement développé par Cisco et qu'on retrouve sur beaucoup de routeurs. GRE a été conçu pour répondre à un maximum de besoins d'encapsulation. Il permet en fait d'encapsuler n'importe quel protocole de niveau 3 dans n'importe quel autre protocole de niveau 3. Puissant... Mais sous Linux, GRE se retrouve sur IP seulement. Reprenons l'architecture décrite en figure 3 et montons notre tunnel GRE. Sur A, nous exécutons :

```
cbr@routerA:~# modprobe ip_gre
cbr@routerA:~# ip tunnel add netb mode gre remote 172.16.2.1 local 172.16.1.1 ttl 255
cbr@routerA:~# ip link set netb up
cbr@routerA:~# ip addr add 10.0.0.1 dev netb
cbr@routerA:~# ip route add 192.168.2.0/24 dev netb
```

Et sur B :

```
cbr@routerA:~# modprobe ip_gre
cbr@routerA:~# ip tunnel add neta mode gre remote 172.16.1.1 local 172.16.2.1 ttl 255
cbr@routerA:~# ip link set neta up
cbr@routerA:~# ip addr add 10.0.0.2 dev neta
```

Et c'est parti. Notons que nous avons donné aux interfaces de tunneling des noms

explicites (neta et netb). GRE permettant en effet de monter plusieurs tunnels, il est bon de savoir sur quel réseau vont déboucher les paquets envoyé par une interface rien qu'en regardant son nom.
GRE supporte aussi l'encapsulation IPv6. Dans ce cas, le tunnel doit être monté en mode SIT :

```
cbr@routerA:~# modprobe ip_gre
cbr@routerA:~# ip tunnel add netb6 mode sit remote 172.16.2.1 local 172.16.1.1 ttl 255
cbr@routerA:~# ip link set netb6 up
```

Ensuite, nous configurons l'interface netb6 avec une adresse IPv6 et routons l'adressage IPv6 du réseau B à travers netb6. Enfantin. Si vous activer le support IPv6, vous aurez aussi la possibilité de faire des tunnels IPv6 sur IPv6 (~~CONFIG_IPV6_TUNNEL~~) conformément à la RFC 2473.
Le tunneling est donc une chose aisée à réaliser au niveau noyau sous Linux. Nous ne parlerons pas des implémentations en espace utilisateur tellement il y en a (nous en avons déjà vu trois s'appuyant sur le pilote TUN/TAP). Il ressort que le mode de tunneling le plus souple est GRE et qu'on le préférera à IPIP, en particulier pour son interopérabilité. Sur un système aussi ouvert que Linux, il serait dommage d'utiliser des modes de communication qui lui sont spécifiques...

IP ARP daemon, TCP Explicit Congestion Notification et TCP syncookie

Trois options à manipuler avec précaution :

- ~~ARP daemon~~ : permet l'utilisation d'un démon ARP en espace utilisateur, ~~arpd~~, via ~~/dev/arpd~~. Pour les réseaux nécessitant beaucoup d'enregistrements ARP, le cache peut devenir très volumineux, consommant une grosse quantité de mémoire noyau, d'où l'idée de déléguer une partie du stockage des entrées à un applicatif. Dans ce cas, le cache ARP se trouve limité à 256 entrées, les suivantes étant gérées par le démon ~~arpd~~. Pratique, mais si jamais le démon n'est pas lancé, votre cache ARP se trouvera de facto limité à une taille inférieure à sa taille habituelle !
- ~~TCP Explicit Congestion Notification~~ : support du TCP ECN, mécanisme permettant aux routeurs de signaler les congestions réseau aux utilisateurs dans le but de limiter les pertes de paquets induites, et donc améliorer les performances du réseau. De nombreux firewalls refusent les connexions de machines ayant cette fonctionnalité activée. Donc, en attendant que leurs administrateurs se soient mis à la page, on devra désactiver le mécanisme :

```
cbr@elendil:~# echo 0 > /proc/sys/net/ipv4/tcp_ecn
```

- ~~TCP syncookio~~ : ce mécanisme apporte une protection contre les dénis de service de type SYN flood. En utilisant un mécanisme de challenge cryptographique connu sous le nom de SYN Cookies, un utilisateur légitime peut accéder à un système "floodé" en outrepassant la mise en file (puisque la file est pleine) par un système de challenge/réponse sur les numéros de séquences initiaux (ISN) des deux protagonistes. La contrepartie étant que, comme une partie de l'ISN sert de challenge, vous réduisiez d'autant la force de votre générateur d'ISN et devenez plus vulnérables à des attaques nécessitant de deviner l'ISN fourni pas le serveur. Pour plus d'information sur les SYN Cookies, voyez la page [14] de DJ Bernstein sur le sujet. En tout état de cause, ce mécanisme est désactivé par défaut, et je vous conseille de le laisser dans cet état, d'autant que Linux encaisse très bien ce type d'attaques. Si l'envie vous prend de l'activer :

```
cbr@elendil:~# echo 1 > /proc/sys/net/ipv4/tcp_syncookies
```

IPSEC, IPv6 et IPVS

La grande nouveauté de la couche réseau des noyaux 2.6 est l'intégration du support IPSEC sous forme de transformations. Certains seront nostalgiques de la gestion par interfaces comme c'était le cas avec FreeS/WAN [15], mais c'est un grand pas en avant que d'avoir, enfin, une implémentation officielle et de qualité d'IPSEC dans le noyau. Le support IPSEC s'active via les deux options que sont AH transformation, ESP transformation. En outre, nous pourrons activer la compression par le choix de IPComp transformation. La configuration de cette couche se fait à l'aide des outils IPSEC [16] portés du projet KAME qui utilisent des sockets spéciales, les PF_KEY, que nous devons donc activer avec l'option ~~PF_KEY sockets~~ au début de la section Networking options. Une interface de configuration spécifique à Linux est également disponible, pour peu que l'option IPsec user configuration interface soit donc activée (et nous l'activerons, évidemment).
Notons que les fonctions cryptographiques nécessaires à la couche IPSEC sont assurées par la CryptoAPI [17] qui est maintenant intégrée aux noyaux officiels. Lorsque nous activons le support IPSEC, la CryptoAPI est activée avec les modules HMAC, MD5, SHA1, DES/3DES (ainsi que le module Deflate si nous choisissons IPComp). Là encore, c'est une avancée certaine de ne plus avoir qu'une couche cryptographique au sein du noyau. Cela va grandement faciliter, par exemple, l'utilisation de périphériques d'accélération cryptographique.
La configuration de la couche IPSEC se fait avec l'outil setkey qui sert à définir les paramètres des associations de sécurité (SA) et des fichiers de configuration. Le noyau supporte les SA manuelles et les SA automatiques. Ces dernières peuvent se faire sur la base d'une clé partagée ou d'un certificat x509, par l'intermédiaire du démon IKE racoon. Pour plus de détails, voir le chapitre 7, IPSEC: secure IP over the Internet, du LARTC [13]. En l'espèce, je vais reprendre l'exemple de la figure 3 pour monter rapidement un tunnel IPSEC manuel. Il s'agit donc de faire

communiquer NetA et NetB via nos deux passerelles A et B. Sur A, nous configurons le tunnel :

```
root@routeurA:~# setkey flush
root@routeurA:~# setkey spdflush
root@routeurA:~# setkey 172.16.1.1 172.16.2.1 esp 300 -m tunnel -E 3des-cbc "123456789012123456
root@routeurA:~# setkey spadd 192.168.1.0/24 192.168.2.0/24 any -P out ipsec esp/tunnel/172.16.1
root@routeurA:~# setkey spadd 192.168.2.0/24 192.168.1.0/24 any -P in ipsec esp/tunnel/172.16.1.
```

Ainsi, nous spécifions d'abord que nous montons un tunnel entre A et B, en utilisant 3DES avec la clé ~~123456789012123456789012~~ et ~~300~~ comme SPI pour ESP. Ensuite, nous disons que le trafic sortant de NetA vers NetB doit passer par le tunnel précédemment défini, et que le trafic en retour de NetB vers NetA doit arriver en utilisant ce même tunnel. Nous ne devons pas oublier de configurer notre filtrage IP pour laisser passer ESP entre A et B (protocole IP 50).

```
root@routeurA:~# iptables -A INPUT -p 50 -s 172.16.2.1 -j ACCEPT
root@routeurA:~# iptables -A OUTPUT -p 50 -d 172.16.2.1 -j ACCEPT
```

Sur B, les règles sont les mêmes, au sens de paquets près :

```
root@routeurB:~# setkey flush
root@routeurB:~# setkey spdflush
root@routeurB:~# setkey 172.16.1.1 172.16.2.1 esp 300 -m tunnel -E 3des-cbc "123456789012123456
root@routeurB:~# setkey spadd 192.168.1.0/24 192.168.2.0/24 any -P in ipsec esp/tunnel/172.16.1.
root@routeurB:~# setkey spadd 192.168.2.0/24 192.168.1.0/24 any -P out ipsec esp/tunnel/172.16.1
root@routeurB:~# iptables -A INPUT -p 50 -s 172.16.1.1 -j ACCEPT
root@routeurB:~# iptables -A OUTPUT -p 50 -d 172.16.1.1 -j ACCEPT
```

La couche IPv6 a été fortement retravaillée, en particulier pour y inclure une couche de sécurité s'appuyant sur IPSEC. Nous avons donc à disposition les supports des extensions de confidentialité définies par la RFC 3041 (assignement périodique de nouvelles adresses à l'interface), de AH, de ESP et d'IPComp. Comme vu précédemment, les tunnels IPv6 sur IPv6 sont également supportés. Autre nouveauté, l'inclusion dans les sources officielles du projet IPVS [18] qui fournit à Linux les capacités nécessaires à la constitution de fermes de serveurs avec partage de charge et haute disponibilité. IPVS supporte le partage de charge pour TCP, UDP, AH et ESP, ainsi que, à l'aide d'un helper spécifique, FTP. Ce projet pourra intéresser tous ceux qui veulent implémenter des firewalls redondants sous Linux avec synchronisation d'état, puisque cette fonctionnalité est disponible pour le ~~load balancer~~ IPVS (director), alors que le projet Netfilter n'a toujours rien d'implémenté à ce niveau-là. Pour plus de détails sur IPVS, voyez la documentation sur le site du projet.

802.1d Ethernet Bridging, Frame Diverter, Network packet filtering et 802.1Q VLAN Support

Intéressons-nous un peu aux fonctions de niveau 2 concernant le support des

ponts Ethernet (conformes à la norme IEEE 802.1d) et le support de l'encapsulation 802.1q pour la transport des VLANs. "Mais que vient faire le filtrage de paquets là-dedans ?" me direz-vous. Je le place ici parce que les nouveautés dans cette section sont l'inclusion dans le noyau du filtrage de paquet au-dessus d'un pont (bridging firewall) et le filtrage de niveau 2.
En activant l'option 802.1d Ethernet Bridging, nous permettons à notre système Linux de se comporter comme un commutateur Ethernet (i.e. un switch) entre plusieurs interfaces physiques. Cette fonctionnalité se configure avec l'outil brctl qu'on trouve dans le ~~paquetage bridge-utils~~ [19] :

```
root@elendil:~# modprobe bridge
root@elendil:~# brctl addbr br0
```

Nous disposons à présent d'une interface, ~~br0~~, matérialisant ce que nous appellerons par la suite un pont (appellation abusive mais commode). Notons qu'une machine peut supporter plusieurs ponts (~~br0~~, ~~br1~~, etc.) mais, en toute logique, une interface physique ne peut appartenir qu'à un seul pont à la fois. Bien que la fonction de pontage ne nécessite aucune configuration IP (fonction de niveau 2), nous pouvons tout de même configurer ~~br0~~ comme n'importe quelle interface classique, rendant ainsi notre machine joignable sur le réseau Ethernet auquel appartient notre pont par une IP. Dans certains cas, comme la mise en place de "firewalls transparents", nous n'affecterons pas d'IP au pont. Nous allons à présent lier des interfaces à notre pont :

```
root@elendil:~# ifconfig eth0 up
root@elendil:~# brctl addif br0 eth0
root@elendil:~# ifconfig eth1 up
root@elendil:~# brctl addif br0 eth1
```

En examinant le log kernel (par dmesg par exemple), nous voyons que nos interfaces passent en mode ~~promiscuous~~ :

```
root@elendil:~# dmesg
[...]
NET4: Ethernet Bridge 008 for NET4.0
Bridge firewalling registered
eth0: Setting promiscuous mode.
device eth0 entered promiscuous mode
eth1: Setting promiscuous mode.
device eth1 entered promiscuous mode
```

En effet, nos interfaces doivent à présent s'intéresser à toutes les trames qu'elles reçoivent, quelle que soit leur adresse MAC destination, pour pouvoir les traiter. Enfin, nous activons notre pont en montant l'interface associée :

```
root@elendil:~# ifconfig br0 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255
```

Si nous ne voulons pas affecter d'IP, un ~~ifconfig br0 up~~ suffira. Les logs confirment que le pont est activé :

```
root@elendil:~# dmesg
[...]
br0: port 2(eth1) entering learning state
br0: port 1(eth0) entering learning state
br0: port 2(eth1) entering forwarding state
br0: topology change detected, propagating
br0: port 1(eth0) entering forwarding state
br0: topology change detected, propagating
```

Notons que l'adresse MAC utilisée par le pont lorsqu'il communique est celle de la première interface qui lui a été associé, à savoir celle mentionnée comme étant le port 1. Notons aussi que ce driver supporte le Spanning Tree (STP) et donc que votre machine Linux peut s'intégrer dans une architecture de commutation Ethernet mettant en œuvre ce protocole. Dans le doute, je vous conseille de le désactiver, juste après avoir créé votre pont :

```
root@elendil:~# brctl stp br0 off
```

L'état de vos ponts et leurs statistiques peuvent être consultées à l'aide de la commande ~~brctl~~ (rien dans ~~/proc/net~~ cette fois-ci) :

```
root@elendil:~# brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.000102402e0a       yes             eth0
                                                        eth1

root@elendil:~# brctl showmacs br0
port no mac addr                is local?       ageing timer
  1     00:01:02:40:2e:0a       yes                0.00
  2     00:01:05:dd:ef:43       yes                0.00
  1     00:02:7e:22:c3:48       no                 3.60
  1     00:0a:b7:58:33:94       no                 0.71
  1     00:10:a4:bb:4a:d3       no                15.97
  2     00:10:a4:bb:8a:51       no                37.30
  1     00:10:b5:81:8a:d2       no               156.98
  1     00:e0:00:59:cb:08       no               279.86
  1     00:e0:00:5c:88:33       no               126.71
root@elendil:~# brctl showstp br0
br0
 bridge id              8000.000102402e0a
 designated root        20ac.0005dc19fc00
 root port                 1                    path cost                104
 max age                  20.00                 bridge max age           20.00
 hello time                2.00                 bridge hello time         2.00
 forward delay            15.00                 bridge forward delay     15.00
 ageing time             300.00                 gc interval               4.00
 hello timer               0.00                 tcn timer                 0.00
 topology change timer     0.00                 gc timer                  3.83
 flags

eth0 (1)
 port id                8001                    state                    forwarding
 designated root        20ac.0005dc19fc00       path cost                100
 designated bridge      80ac.000ab7583380       message age timer          1.67
 designated port        8014                    forward delay timer        0.00
 designated cost           4                    hold timer                 0.00
```

```
 flags

eth1 (2)
 port id                8002                    state                   forwarding
 designated root        20ac.0005dc19fc00       path cost                100
 designated bridge      8000.000102402e0a       message age timer          0.00
 designated port        8002                    forward delay timer        0.00
 designated cost         104                    hold timer                 0.67
 flags
```

L'option Frame Diverter permet à un pont de rediriger vers lui des trames au lieu de les "forwarder". C'est l'équivalent au niveau Ethernet de la cible ~~REDIRECT~~ de Netfilter. Cette option nous permet donc de mettre en place des proxies transparents directement au niveau 2. À titre d'exemple succinct, nous allons configurer un proxy HTTP transparent au moyen de Squid [20] sur une machine possédant un pont ~~br0~~ sur ses deux interfaces ~~eth0~~ et ~~eth1~~. Tout d'abord, Squid doit être configuré pour supporter le mode transparent. Nous le plaçons en écoute sur le port 3128 :

```
http_port 3128
httpd_accel_host virtual
httpd_accel_port 80
httpd_accel_with_proxy  on
httpd_accel_uses_host_header on
```

Le diverter se configure avec l'outil ~~divert~~ disponible sur le site [21] dédié à cette fonctionnalité. Si vous voulez utiliser la version fournie avec le noyau, à savoir la version stable 0.46, il vous faudra utiliser la version 0.221 de ~~divert~~. Si vous voulez utiliser la toute dernière version, la 0.52, vous devrez utiliser divert 0.32. Non seulement, les versions ne sont pas compatibles, mais en outre, elles n'ont pas la même syntaxe (le numéro de la version compilée dans le noyau courant est contenue dans le fichier ~~/proc/sys/net/core/divert_version~~). Nous utiliserons ici la version du noyau officiel.
Nous allons remonter au niveau IP les trames contenant les requêtes HTTP qui nous intéressent. L'interface considérée est celle par laquelle les requêtes vont arriver au pont. Nous redirigeons donc les trames contenant un segment TCP à destination du port 80 que nous recevons sur l'interface ~~eth0~~ :

```
root@elendil:~# divert on eth0 enable
root@elendil:~# divert on eth0 tcp add dst 80
```

Nous pouvons à tout moment vérifier notre configuration courante en appelant la commande sur l'interface considérée sans argument :

```
root@elendil:~# divert on eth0
version: 0.46
status:  active
ip:      no
icmp:    no
tcp:
         * -> 80
```

```
udp:      no
```

Une fois les paquets IP contenus de ces trames redirigés vers la couche IP, nous n'avons plus qu'à les rediriger vers notre proxy Squid avec une simple règle de NAT :

```
root@elendil:~# iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports 3128
```

Notons qu'il y a maintes façons de mettre en place un proxy transparent, que ce soit en utilisant un routage avancé à base de marques ou encore une NAT sur la destination des requêtes HTTP. Ceci en est une parmi d'autres, que je trouve plutôt élégante quand on utilise un pont.

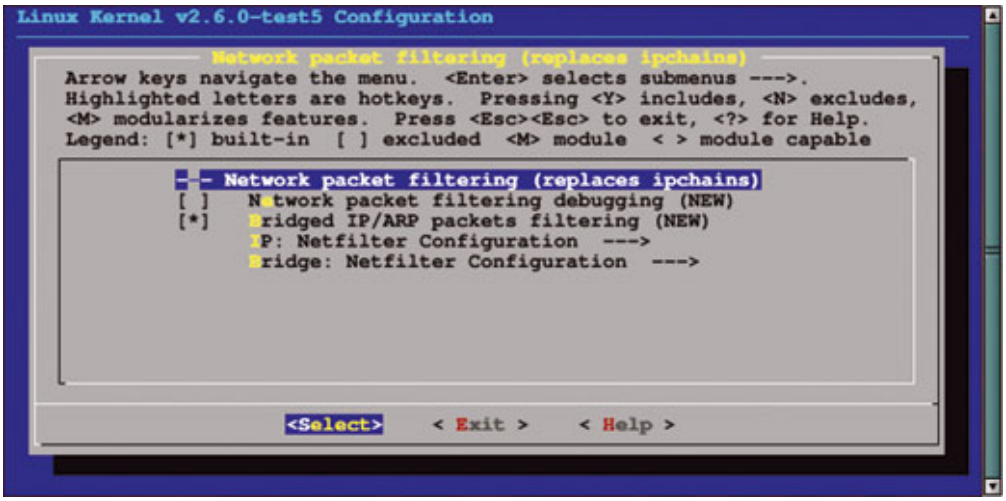

Fig. 4 : Menu de configuration du filtrage de paquets.

Une fois l'option ~~bridging~~ activée, nous constatons que deux nouvelles entrées sont présentes dans le sous-menu de configuration du filtrage de paquets, Network packet filtering, comme le montre la figure 4 :

- ~~Bridged IP/ARP packets filtering~~ : cette option permet de voir apparaître dans la chaîne FORWARD de Netfilter la charge des trames traitées par notre pont si celle-ci est de l'ARP ou de l'IP. Nous pouvons donc filtrer ces paquets, créant ainsi un pont filtrant, base des "firewalls transparents". Le principe est fort simple et si vous désirez en savoir plus sur cette fonctionnalité, je vous conseille la lecture d'un article [22] sur le sujet paru dans le Hors Série 13 de Linux Magazine France.
- ~~Bridge: Netfilter Configuration~~ : cette option permet de configurer le filtrage de niveau 2 au niveau de vos ponts (et seulement sur des ponts !) à l'aide de l'outil ~~ebtables~~ [23]. Ce dernier vous permettra de filtrer les trames sur la base de leurs en-têtes de niveau 2 (adresses MAC, type de la charge, etc.) pour différents protocoles (Ethernet, 802.3, 802.1q et STP), un filtrage complet des paquets ARP, un filtrage simple sur quelques en-têtes IP, du mangling (dont le marquage de trames) et du NAT. Pour ARP et IP en

particulier, ~~ebtables~~ vous permettra d'imposer des règles de validation MAC/IP non négligeables dans certains environnements.

Du côté de Netfilter proprement dit, il n'y a pas grand-chose de nouveau sous le soleil au niveau de l'architecture générale. Je vous renvoie donc à mon article [24] sur le sujet paru dans le Hors Série 12. Côté fonctionnalités, de nouvelles concordances et cibles ont été introduites. Enfin, le support du protocole ARP (filtrage et mangle) dispose maintenant d'un outil de configuration, arptables, disponible sur le site de ebtables [23].
L'option 802.1Q VLAN Support permet à notre système de supporter le protocole de transport de VLAN par encaspulation (IEEE 802.1q) sur des liens couramment appelés trunks (terminologie Cisco). Comme le montre la figure 5, la trame 802.1q se différencie de la trame Ethernet classique par l'insertion d'un champ de 2 octets permettant d'une part d'identifier le VLAN auquel appartient la trame (champ VLAN ID sur 12 bits) et d'autre part de lui affecter une priorité sur le trunk (champ Prio), et d'un nouveau champ Type (le premier désigne la trame comme trame 802.1q, le second définissant le type de charge).

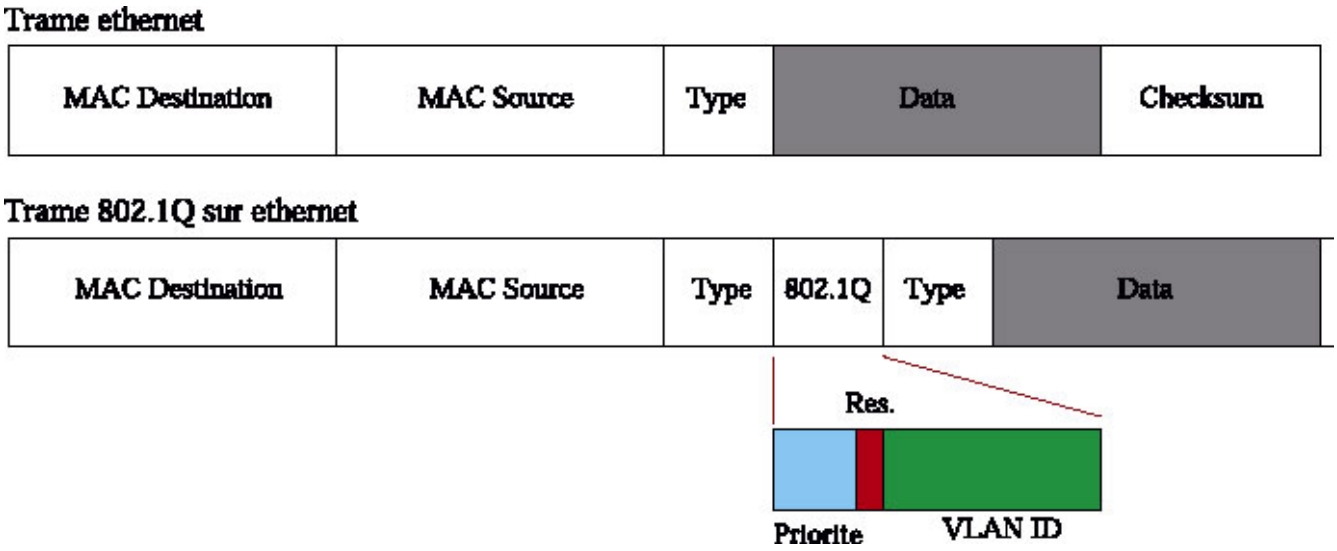

Fig. 5 : Trame Ethernet et trame 802.1q.

En reliant une interface Ethernet de notre système à un commutateur supportant l'encapsulation 802.1q, nous sommes capable de recevoir plusieurs VLAN. Pour notre couche réseau, chaque VLAN sera vu comme une interface logique. La configuration 802.1q se fait avec l'outil vconfig disponible sur le site [25] dédié au support des VLANs. Si nous voulons configurer notre machine comme décrit en figure 6, nous procéderons comme suit. Commençons donc par configurer nos VLANs sur l'interface destinée à les recevoir :

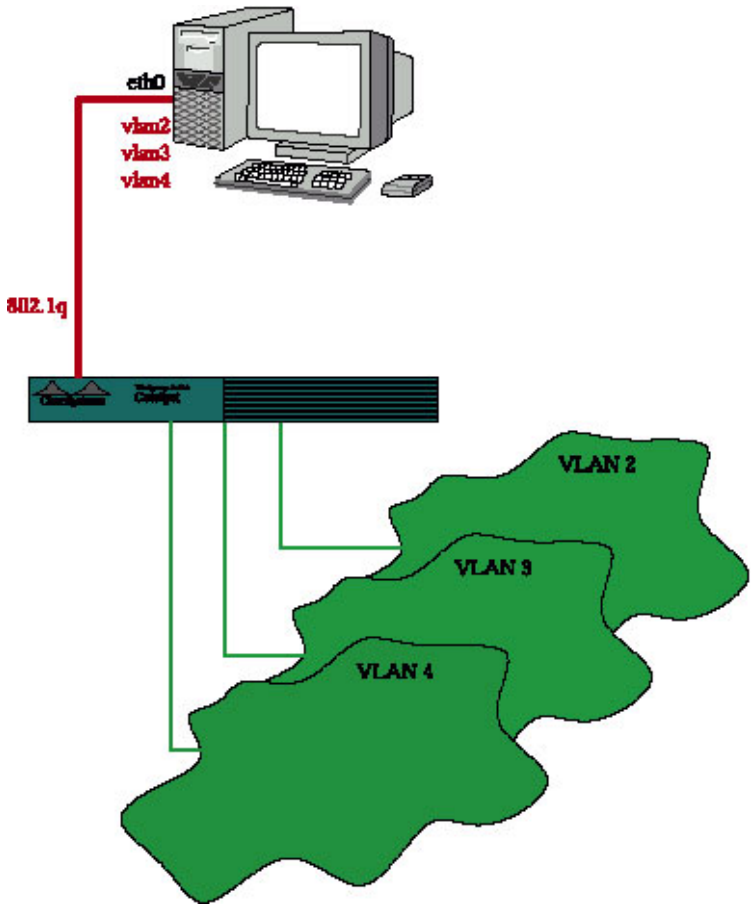

Fig. 6 : Architecture à base de VLANs.

```
root@elendil:~# modprobe 8021q
root@elendil:~# ifconfig eth0 up
root@elendil:~# vconfig set_name_type VLAN_PLUS_VID_NO_PAD
Set name-type for VLAN subsystem. Should be visible in /proc/net/vlan/config
root@elendil:~# vconfig add eth0 2
Added VLAN with VID == 2 to IF -:eth0:-
root@elendil:~# vconfig add eth0 3
Added VLAN with VID == 3 to IF -:eth0:-
root@elendil:~# vconfig add eth0 4
Added VLAN with VID == 4 to IF -:eth0:-
root@elendil:~# vconfig add eth0 5
Added VLAN with VID == 5 to IF -:eth0:-
```

Nous n'utilisons pas le VLAN 1 à dessein. En effet, ce VLAN présente chez certains constructeurs (Cisco en particulier) des particularités qui interdisent son utilisation à des fins de transport de trafic réseau, le réservant à un usage uniquement administratif. On obtient donc sur eth0 ce qu'on appelle un trunk, un lien destiné à transporter des VLANs, c'est-à-dire des trames 802.1q. La commande ~~vconfig set_name_type VLAN_PLUS_VID_NO_PAD~~ permet de choisir le modèle de nommage des interfaces logiques associées aux différents VLANs. Quatre modèles sont disponibles :

- VLAN_PLUS_VID : vlan suivi du numéro de VLAN sur quatre chiffres, i.e. vlan0005 ;
- VLAN_PLUS_VID_NO_PAD : vlan suivi du numéro court de VLAN, i.e. vlan5 (c'est ce que j'ai choisi ici) ;
- DEV_PLUS_VID : interface suivie du numéro de VLAN sur quatre chiffres, i.e. eth0.0005 ;
- DEV_PLUS_VID_NO_PAD : interface suivie du numéro court de VLAN, i.e. eth0.5.

Chaque notation a son intérêt. Si on n'a qu'un seul trunk sur la machine, on utilisera plutôt les notations VLAN_PLUS_*, alors que les DEV_PLUS_* seront clairement adaptées aux configuration à plusieurs trunks. Si on a peu de VLANs, i.e. moins de 10, on utilisera les *_VID_NO_PAD, alors qu'avec un nombre plus important, la notation *_VID rendra un résultat plus clair lorsqu'on listera les interfaces avec ifconfig. Vous aurez remarqué que dans mon élan, j'ai configuré un VLAN de trop (le 5). C'est juste pour vous montrer comment le supprimer ;)

```
root@elendil:~# vconfig rem vlan5
Removed VLAN -:vlan5:
```

Une fois votre trunk mis en place, nous pouvons allez jeter un coup d'œil dans /proc/ pour en regarder la configuration ainsi que les statistiques de chaque VLAN :

```
root@elendil:/proc/net/vlan# ls
config  vlan2  vlan3  vlan4
root@elendil:/proc/net/vlan# cat config
VLAN Dev name    | VLAN ID
Name-Type: VLAN_NAME_TYPE_PLUS_VID_NO_PAD
vlan2          | 2  | eth0
vlan3          | 3  | eth0
vlan4          | 4  | eth0
root@elendil:/proc/net/vlan# cat vlan2
vlan2  VID: 2    REORDER_HDR: 1  dev->priv_flags: 1
         total frames received:            0
          total bytes received:            0
      Broadcast/Multicast Rcvd:            0

      total frames transmitted:            0
       total bytes transmitted:            0
            total headroom inc:            0
           total encap on xmit:            0
Device: eth0
INGRESS priority mappings: 0:0  1:0  2:0  3:0  4:0  5:0  6:0 7:0
EGRESSS priority Mappings:
```

Bien entendu, vous devez disposer en face d'un commutateur configuré en conséquence, ce qui, sur un Cisco 2950 [2] nous donne quelque chose dans ce genre-là (mode trunk pur, sans négociation possible) :

```
2950# configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
```

```
2950(config)# interface fastethernet0/1
2950(config-if)# switchport mode trunk
2950(config-if)# switchport nonegotiate
2950(config-if)# switchport trunk native vlan 100
2950(config-if)# switchport trunk allowed vlan add 2-4
2950(config-if)# end
```

Notons qu'il est tout à fait possible de créer un trunk au-dessus d'un agrégat, aussi bien du côté Linux que du côté du commutateur Ethernet. Il est intéressant de considérer cette éventualité si on ne dispose pas de carte GigabitEthernet dans la mesure où les trunks sont par essence des liens très chargés sur une architecture commutée. Notons aussi que nous pouvons toujours émettre ou recevoir des trames Ethernet classiques en utilisant directement eth0, mais que ce type d'utilisation (lien hybride) est clairement déconseillée.
Et maintenant que vos interfaces sont en place, vous pouvez activer le routage entre les différents VLANs et configurer un filtrage de paquets ou de la gestion de bande passante, bref, tout ce que vous pouviez faire de vos interfaces physiques. Au sein d'une architecture réseau, l'utilisation des VLANs vous permet de rationaliser l'utilisation des ports de vos commutateurs Ethernet même lorsque le nombre de réseaux IP augmente. En outre, au niveau d'un pare-feu, vous pouvez multiplier les DMZ à moindre coût et donc augmenter votre niveau de segmentation. Il convient cependant d'appréhender le concept des VLANs avec prudence dans le cadre d'une architecture de sécurité, en ce que le mécanisme des VLANs reste une solution logicielle qui peut souffrir des bogues. En outre, faire du firewalling sur un trunk revient à déléguer une partie du rôle de pare-feu au commutateur Ethernet dans la mesure où il décide dans quel VLAN placer ses trames, et donc sur quelle interface du firewall arrivera le paquet correspondant.

QoS and/or fair queueing

La toute dernière fonctionnalité intéressante, et non la moindre, est la gestion de la qualité de service, tant au niveau de la bande passante que de la priorité des flux. Nous avons précédemment vu le module shaper qui nous permettait de limiter de la bande passante de manière fort simple. Les méthodes que nous allons voir à présent sont nettement plus puissantes, mais aussi plus difficiles à appréhender.
Globalement, la section QoS and/or fair queueing est divisée en trois groupes :

- Les ordonnanceurs (~~schedulers~~), chargés de la gestion des files d'attente (~~queue~~) pour une interface donnée. Nous disposons de divers algorithmes dont les deux plus courants sont CBQ (~~Class Based Queueing~~) et HTB (Hierarchical Token Buckets). Ces deux ordonnanceurs permettent de créer une hiérarchie de files d'attentes avec des notions de parenté et d'héritage.
- Les files d'attentes, qui sont chargées d'émettre les paquets selon un comportement bien spécifique à chaque type. Les deux types le plus souvent retrouvés sont SFQ (~~Stochastic Fairness Queueing~~) et TBF (~~Token Bucket~~

~~Filter~~)
- Les classificateur (~~classifiers~~), sous Packet classifier API, dont le rôle est de ventiler les paquets entre les classes (cf ci-après) disponibles. Les plus utilisés sont U32, qui permet de "matcher" les en-têtes des paquets à l'aide de masques hexadécimaux ; Route, qui permet de trier en fonction de la route suivie ; et enfin Firewall, qui s'appuie sur la reconnaissance de la sacro-sainte marque Netfilter. On peut aussi filtrer en fonction de la valeur du champ TOS, puisque la QoS est sa raison d'être.

Ce qu'il faut bien comprendre, c'est le concept de ~~queueing discipline (qdisc)~~, ou gestion de queue. Les ordonnanceurs et les files d'attentes sont des qdiscs, à savoir des méthodes d'envoi des paquets. Seulement, les ordonnanceurs diffèrent des files en ce qu'ils peuvent avoir des enfants. Ils servent donc à envoyer des paquets vers ces enfants (classes). C'est pourquoi on les appelles ~~classful qdisc~~. Les autres sont des ~~classless qdisc~~, qui ne peuvent pas avoir d'enfant et servent à envoyer les paquets sur le réseau. De fait, une classless qdisc est toujours terminale. En outre, une classless qdisc a toujours un parent, alors que la classful qdisc peut-être une racine.
En outre, nous disposons d'un quatrième concept, celui de classes. Une classe sert à imposer la limitation désirée sur les paquets qui y sont envoyés. Une classe a un parent, qui peut soit être une classful qdisc, soit une autre classe. De fait, une classe peut avoir comme enfant un classe, mais aussi une qdisc. Une classe sans enfant explicite se voit attaché une qdisc de type FIFO. Au final, on obtient un arbre qui commence par une qdisc dite racine de type classful (HTB ou CBQ dans la plupart des cas). Cette qdisc est parente de classes qui elles-mêmes peuvent ou non avoir des enfants. Certaines n'en ont pas, d'autres ont des classes comme filles, d'autres, enfin, sont parentes de qdisc de type classless. Nous disposons en parallèle d'une série de classificateurs dont le rôle est de ventiler les paquets dans les différentes classes.
Le fonctionnement est le suivant. Une qdisc est une manière d'envoyer les paquets qui sont dans une file d'attente. La qdisc racine a pour rôle d'envoyer les paquets vers les classificateurs qui vont les ventiler les classes filles. Celles-ci vont alors devoir envoyer leurs propres paquets, soit à une autre classe (auquel cas on parcourt l'arbre en utilisant un autre classificateur), soit sur l'interface en utilisant une qdisc classless, FIFO par défaut ou la qdisc imposée par l'utilisateur (généralement TBF ou SFQ).
Si vous comptez aller plus loin et vous tourner vers la qualité de service avec priorités sur les flux et réservation de bande passante, il vous faudra activer les options QoS support et Rate estimator. Cette dernière permet au noyau de mesurer le trafic présent sur une interface donnée et donc d'adapter la qualité de service en fonction ; c'est une fonctionnalité indispensable à une bonne implémentation de la QoS. Vous aurez à votre disposition un classificateur RSVP (Resource Reservation Protocol) et pourrez utiliser DiffServ (Differentiated Services). Pour plus de détails, je vous renvoie vers le site DiffServ pour Linux [26], véritable mine d'informations sur le sujet. Enfin, activez l'option Traffic policing.

Notez bien que la gestion de la bande passante est largement dédiée au trafic sortant. Si vous voulez gérer du trafic entrant, vous devez sélectionner l'option Ingress Qdisc. Une autre fonctionnalité sympathique, disponible sous forme de patch, est IMQ (Intermediate queueing device) [27]. IMQ vous fournit une cible Netfilter dont le rôle est d'envoyer les paquets vers une interface (imq0) directement attaché à un ordonnanceur. Vous pouvez ainsi traiter n'importe quel type de paquet, entrant ou sortant, mais surtout choisir quels paquets vont passer par la gestion de QoS et quels paquets qui n'y passeront pas.
Dans la pratique, tout ce beau monde se configure avec l'outil tc fourni dans le paquetage iproute2 [12]. Ce dernier vous servira à créer votre arbre, y affecter vos files et configurer les classificateurs. La démarche est plutôt longue et fastidieuse. Considérons l'exemple très simple de la limitation de la bande passante pour un seul hôte, dont le trafic sort par l'interface ~~eth0~~ de notre système :

```
tc qdisc add dev eth0 root handle 1: cbq avpkt 1000 bandwidth 100mbit
tc class add dev eth0 parent 1: classid 1:1 cbq rate 512kbit allot 1500 prio 5 bounded isolated
tc filter add dev eth0 parent 1: protocol ip prio 16 u32 match ip src 192.168.1.10 flowid 1:1
```

Nous devons créer une racine pour l'ordonnanceur (~~tc qdisc add~~). Nous greffons sur cette racine une classe fille limitée à 512kbps (~~tc class add~~). Enfin, nous configurons notre classificateur pour qu'il envoie sur cette dernière classe les paquets en provenance de 192.168.1.10 (~~tc filter add~~). Et ceci ne concerne qu'un seul sens. Si je veux mettre en place une limitation bidirectionnelle, il faut mettre un jeu de règles similaires sur ~~eth1~~ :

```
tc qdisc add dev eth1 root handle 1: cbq avpkt 1000 bandwidth 100mbit
tc class add dev eth1 parent 1: classid 1:1 cbq rate 512kbit allot 1500 prio 5 bounded isolated
tc filter add dev eth1 parent 1: protocol ip prio 16 u32 match ip dst 192.168.1.10 flowid 1:1
```

Si en plus, je veux utiliser une gestion de file d'attente particulière plutôt qu'une simple file FIFO (comportement par défaut), je le précise ainsi :

```
tc qdisc add dev eth0 parent 1:1 handle 10: sfq perturb 10
tc qdisc add dev eth1 parent 1:1 handle 10: sfq perturb 10
```

De cette manière, les paquets qui vont passer par la classe ~~1:1~~ limitée à 512kbps seront traités par le mécanisme SFQ. On peut vérifier le tout en demandant à tc de nous afficher les paramètres en cours sur eth0 par exemple :

```
root@elendil:/etc/cbq# tc qdisc show dev eth1
qdisc sfq 10: quantum 1514b perturb 10sec
qdisc cbq 1: rate 100Mbit (bounded,isolated) prio no-transmit
root@elendil:/etc/cbq# tc class show dev eth1
class cbq 1: root rate 100Mbit (bounded,isolated) prio no-transmit
class cbq 1:1 parent 1: leaf 10: rate 512Kbit (bounded,isolated) prio 5
root@elendil:/etc/cbq# tc filter show dev eth1
filter parent 1: protocol ip pref 16 u32
filter parent 1: protocol ip pref 16 u32 fh 800: ht divisor 1
filter parent 1: protocol ip pref 16 u32 fh 800::800 order 2048 key ht 800 bkt 0 flowid 1:1
  match c0a8010a/ffffffff at 16
```

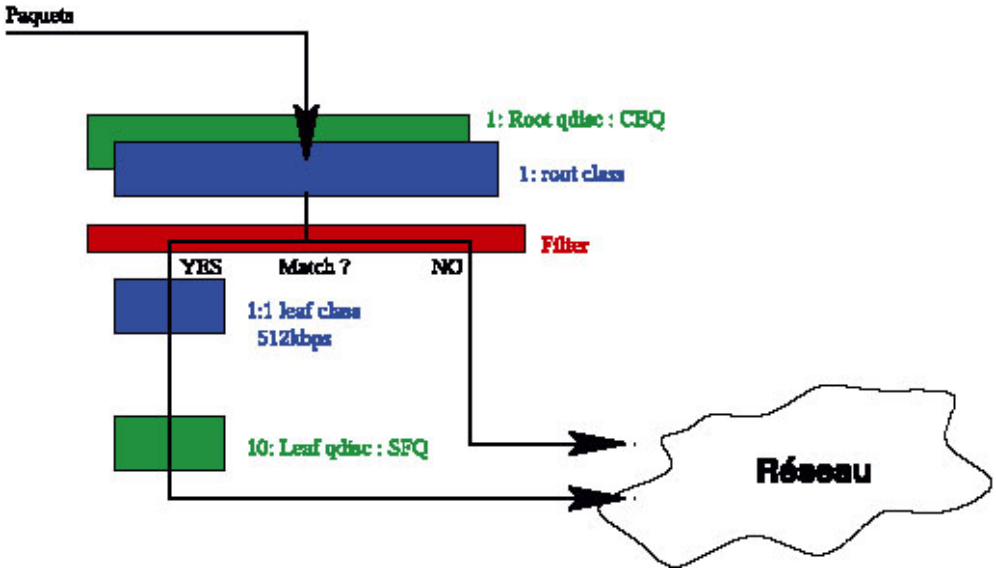

Fig. 7 : Configuration CBQ simple.

Voyons la figure 7. Nous avons une qdisc racine nommée 1: de type CBQ (classful) et une qdisc 10: de type SFQ (classless). 1: a une classe fille également appelée 1:. Celle-ci est créée automatiquement et prend toute la bande passante disponible sur la qdisc parente puisqu'elle en traitera tous les paquets. La classe 1:1 a pour parente la classe 1: (parent 1:) et pour fille la qdisc classless 10: (leaf 10:) de type SFQ. Les paquets qui passeront par elle sont choisis par le troisième filtre (flowid 1:1), qui est attaché à la classe 1: (parent 1:). La bande passante de 1:1 est limitée à 512kbps et les paquets sont envoyés en utilisant la qdisc SFQ (10:). Le choix entre SFQ et TBF est relativement simple. TBF vous permet d'obtenir une limitation très précise du trafic, mais ne fait pas de distinction entre les flux qui le composent, pouvant conduire à la monopolisation de la bande passante par une application au détriment des autres. SFQ est moins précise, mais elle distingue les flux de niveau 4 (typiquement UDP et TCP) et tente de les traiter de manière égale. Ainsi, on ne risque pas de voir une application prendre le dessus et gêner les autres. La classe 1:1 est enfin bornée (bounded), ce qui signifie qu'elle ne peut emprunter de bande passante à sa classe parente lorsqu'elle atteint sa limite, même si cette dernière dispose de bande passante libre, et isolée (isolated), ce qui signifie qu'elle ne peut pas prêter de bande passante disponible à des sœurs qui ne seraient pas elles-mêmes bornées. Les concepts de classe bornée et de classe isolée nous permettent de mettre en place des schémas de gestion de bande passante plus dynamiques avec, par exemple, la notion de bande passante minimum.
Je ne vous cache pas que dès que vous avez besoin de spécifier plusieurs politiques de limitations, les choses commencent à devenir de plus en plus compliquées, comme illustré en figure 8. Heureusement, nous avons à notre disposition deux scripts qui nous permettent de concevoir des architectures de gestion de trafic de manière raisonnablement simple et souple. Le premier, CBQ.init [28], s'appuie sur la qdisc CBQ tandis que le second, HTB.init [29],

s'appuie plutôt sur HTB. Ils supportent tous les deux les classificateurs U32, Route et Firewall, et les qdisc classless TBF et SFQ. En ce qui concerne leur configuration, le mode d'emploi figure en commentaire au début des scripts eux-mêmes. So, read the source, Luke, read the source. Les scripts utilisent des fichiers de configuration très simples, de format très proche. Si on sait se servir de l'un, il ne faut guère de temps pour apprendre à utiliser l'autre.

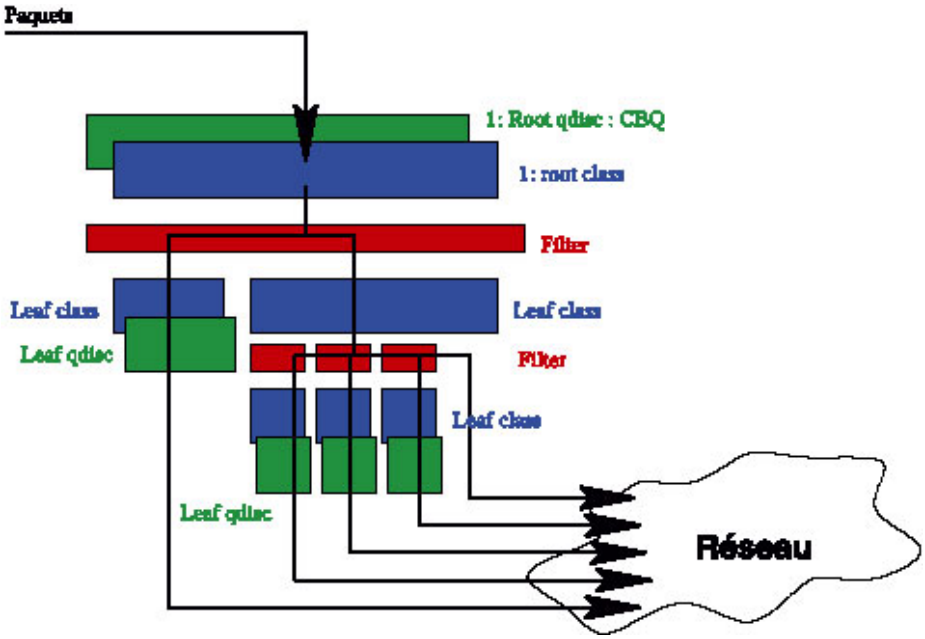

Fig. 8 : Configuration CBQ étoffée.

On saisit bien toute la puissance de la gestion de QoS sous Linux, mais aussi toute la difficulté qui l'accompagne. Même si le travail se trouve facilité par l'utilisation des scripts cités, les concepts sous-jacents sont tout de même (en tout cas pour moi) un peu ardus à appréhender au début. Je vous conseille donc vivement la lecture du chapitre 9, ~~Queueing Disciplines for Bandwidth Management~~, de l'incontournable LART [13] avant de vous lancer dans l'aventure.

Autres fonctionnalités

De nombreuses fonctionnalités en relation avec le réseau sont disséminées dans d'autres sections de l'interface de configuration du noyau. On pourra citer :

- ~~Network block device~~ dans la section Block devices, qui a fait l'objet d'un article [30] dans Linux Magazine ;
- ~~Ethernet over 1394~~ dans la section IEEE 1394 (FireWire), qui vous permettra de créer un lien réseau haut débit sur FireWire ;
- ~~Amateur Radio support~~, qui vous permet de faire du X25 avec des

équipements radio-amateur ;
- ~~IrDA (infrared)~~, qui possède une extension IrLAN vous permettant de communiquer via votre interface infrarouge (en plus du PPP sur IrCOMM) ;
- ~~ISDN support~~ pour le support des modem RNIS ;
- le support des systèmes de fichiers en réseau avec NFS, SMB, CIFS, NCP, Coda, InterMezzo et AFS ;
- ~~Multi-purpose USB Networking Framework~~ dans la section USB qui vous fournit la possibilité de faire du réseau sur un câble USB adéquat, en plus de tous les contrôleurs réseau et modems USB supportés disponibles dans la même section ;
- et enfin la possibilité de monter des réseau Ethernet sur BlueTooth avec le support BNET.

J'ajouterai à tout cela le support des LSM ou (~~Linux Security Modules~~), qui font l'objet d'un article [31] dans ce même numéro, pouvant induire de nouvelles fonctions de sécurité sur l'accès des applications à la couche réseau, en particulier à travers l'option ~~Socket and Networking Security Hooks~~.
Au tout début de cet article, je vous conseillais l'activation du support du Sysctl. Nous avons en effet pu voir qu'il pouvait servir à configurer certaines options ou à récupérer des données (statistiques, état, etc.). Je ne peux pas vous faire ici la liste exhaustive détaillée de toutes les options proposées par le Sysctl pour configurer le comportement de votre machine vis-à-vis du réseau. C'est pourquoi je vous renvoie au chapitre 13, Kernel network parameters, du LARTC [13].
Enfin, les capacités réseau de Linux ne s'arrêtent pas à ce que supporte le noyau Linux. En effet, de nombreuses fonctionnalités sont implémentées sous forme d'applications, parmi lesquelles ont pourra citer le support complet du routage dynamique (RIP, OSPF ou BGP) avec Zebra [32], ou encore la gestion du failover en réseau avec VRRP en utilisant KeepAlived [33].

Conclusion

J'espère vous avoir montré, sinon démontré, au travers de ce descriptif sommaire, l'extrême versatilité du noyau Linux dans l'univers des réseaux. Il peut se connecter à pratiquement tous les réseaux existants ou ayant existé, communiquer en utilisant une large palette de protocoles, s'interfacer avec de nombreux systèmes et enfin remplir un nombre de fonctions particulièrement important. Cela fait de Linux un outil d'une incroyable utilité pour tous ceux qui veulent jouer avec les réseaux et expérimenter des fonctions normalement réservées aux gros systèmes, et, à l'occasion, faire pâlir de jalousie vos collègues devant votre portable que vous pouvez configurer en pont/routeur/firewall/shaper en trois minutes chrono en main.
~~P.S.~~ : Je n'ai malheureusement pas pu me procurer le switch Cisco 2950 pour valider les exemples fournis que je donne de mémoire, avec le manuel sur les genoux ; je réclame donc toute votre indulgence en la matière.

Références

[1] Linux Channel Bonding, http://sourceforge.net/projects/bonding/
[2] Cisco Catalyst 2950 Series Switches Documentation, http://www.cisco.com/en/US/products/hw/switches/ps628/index.html
[3] Tunneling over UDP using tun/tap, http://www.cartel-securite.fr/pbiondi/projects/tuntap_udp.html
[4] VTun, http://vtun.sourceforge.net/
[5] OpenVPN, http://openvpn.sourceforge.net/
[6] Universal TUN/TAP Driver, http://vtun.sourceforge.net/tun/
[7] PPP, http://www.samba.org/ppp/
[8] RoaringPenguin PPPoE, http://www.roaringpenguin.com/pppoe/
[9] Libpcap, http://www.tcpdump.org/
[10] Libnet Packet Assembly, http://www.packetfactory.net/projects/libnet/
[11] Netfilter, http://www.netfilter.org/
[12] iproute2, ftp://ftp.inr.ac.ru/ip-routing/
[13] Bert Hubert et autres auteurs, Linux Advanced Routing and Traffic Control HOWTO, http://lartc.org/
[14] DJ Bernstein, SYN Cookies, http://cr.yp.to/syncookies.html
[15] FreeS/WAN, http://www.freeswan.org/
[16] IPsec-Tools, http://ipsec-tools.sourceforge.net/
[17] The GNU/Linux CryptoAPI site, http://www.kerneli.org/
[18] Linux Virtual Server Project, http://www.linuxvirtualserver.org/
[19] Linux Ethernet Bridging, http://bridge.sourceforge.net/
[20] Squid Web Proxy Cache, http://www.squid-cache.org/
[21] Frame Diverter, http://diverter.sourceforge.net/
[22] Loïc Minier, Un bridge "firewallant", Linux Magazine France HS13
[23] ebtables, http://ebtables.sourceforge.net/
[24] Cédric Blancher, Netfilter/iptables, Linux Magazine France HS12
[25] 802.1Q VLAN implementation for Linux, http://www.candelatech.com/~greear/vlan.html
[26] Differentiated Services on Linux, http://diffserv.sourceforge.net/
[27] The intermediate queueing device, http://trash.net/~kaber/imq/
[28] CBQ.init, http://sourceforge.net/projects/cbqinit/
[29] HTB.init, http://sourceforge.net/projects/htbinit/
[30] Yves Bailly, NBD, http://www.linuxmag-france.org/LINU_ndb.pdf et Linux Magazine France 54
[31] Philippe Biondi et Frédéric Raynal, Les LSM du Futur, Linux Magazine France HS 17
[32] Zebra, http://www.zebra.org/
[33] KeepAlived, http://keepalived.sourceforge.net/

~~Retrouvez cet article dans :~~ Linux Magazine Hors série 17

Posté par (La rédaction) | Article paru dans 

Laissez une réponse

Vous devez avoir ouvert une session pour écrire un commentaire.

« Précédent Aller au contenu »

Identifiez-vous
Inscription
S'abonner à UNIX Garden

Articles de 1ère page

- Noël 94 : le cas Mitnick-Shimomura ou comment le cyber-criminel a souhaité joyeux Noël au samurai
- Mettre en place une politique SSI : des recettes pratiques
- Extensions de Firefox : notre sélection
- Konversation : pour discuter librement sur IRC
- BitTorrent : l'autre façon d'échanger des fichiers
- Au-delà de Diffie-Hellman ... ?
- Envy : l'installation facile des drivers graphiques dernier cri pour ATI et Nvidia
- KAudioCreator : logiciel d'extraction de CD et d'encodage audio
- Les flux réseau
- Clamav, l'antivirus qui vient du froid

Actuellement en kiosque :

- **Il y a actuellement**
- 740 articles/billets en ligne.

Recherche

- **Catégories**

- Administration réseau
- Administration système
- Agenda-Interview
- Audio-vidéo
- Bureautique
- Comprendre
- Distribution
- Embarqué
- Environnement de bureau
- Graphisme
- Jeux
- Matériel
- News
- Programmation
- Réfléchir
- Sécurité
- Utilitaires
- Web

- **Archives**

- septembre 2008
- août 2008
- juillet 2008
- juin 2008
- mai 2008
- avril 2008
- mars 2008
- février 2008
- janvier 2008
- décembre 2007
- novembre 2007
- février 2007

GNU/Linux Magazine

- GNU/Linux Magazine 108 - Septembre 2008 - Chez votre marchand de journaux
- Edito : GNU/Linux Magazine 108
- GNU/Linux Magazine HS 38 - Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : GNU/Linux Magazine HS 38
- GNU/Linux Magazine 107 - Juillet/Août 2008 - Chez votre marchand de journaux

GNU/Linux Pratique

- Linux Pratique N°49 -Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : Linux Pratique N°49
- À télécharger : Les fichiers du Cahier Web de Linux Pratique n°49
- Linux Pratique Essentiel N°3 - Août/Septembre 2008 - Chez votre marchand de journaux
- Edito : Linux Pratique Essentiel N°3

MISC Magazine

- Misc 39 : Fuzzing - Injectez des données et trouvez les failles cachées - Septembre/Octobre 2008 - Chez votre marchand de journaux
- Edito : Misc 39
- MISC 39 - Communiqué de presse
- Salon Infosecurity & Storage expo - 19 et 20 novembre 2008.
- Misc 38 : Codes Malicieux, quoi de neuf ? - Juillet/Août 2008 - Chez votre marchand de journaux