Linux 2.4 Advanced Routing HOWTO

Netherlabs BV (bert hubert <bert.hubert@netherlabs.nl>)
Gregory Maxwell <greg@linuxpower.cx>
Remco van Mook <remco@virtu.nl>
Martijn van Oosterhout <kleptog@cupid.suninternet.com>
Paul B Schroeder <paulsch@us.ibm.com>
howto@ds9a.nl
Traduction de Laurent Foucher <foucher@gch.iut-tlse3.fr>, Philippe Latu <philippe.latu@linux-france.org> et Guillaume Allègre <Guillaume.Allegre@imag.fr>
v0.1.0 $Date: 01/07/2000 12:20:21 $

Une approche pratique d'iproute2, de la mise en forme du trafic et un peu de netfilter.

Table des matières

1 Dédicace

Ce document est dédié à beaucoup de gens ; dans ma tentative de tous me les rappeler, je peux en citer quelques-uns :

- Rusty Russel
- Alexey N. Kuznetsov
- La fine équipe de Google
- L'équipe de Casema Internet

2 Introduction

Bienvenue, cher lecteur.

Ce document a pour but de vous éclairer sur la manière de faire du routage avancé avec Linux 2.2/2.4. Méconnus par les utilisateurs, les outils standard de ces noyaux permettent de faire des choses spectaculaires. Les commandes comme "route" et "ifconfig" sont des interfaces vraiment pauvres par rapport à la grande puissance potentielle d'iproute2.

J'espère que cet HOWTO deviendra aussi lisible que le très réputé Netfilter, de Rusty Russel.

Vous pouvez nous contacter en nous écrivant à *équipe HOWTO* `<mailto:HOWTO@ds9a.nl>`.

2.1 Conditions de distribution et Mise en garde

Ce document est distribué avec l'espoir qu'il sera utile et utilisé. Sauf mention contraire, les documents Linux HOWTO sont la propriété de leurs auteurs respectifs. Les documents Linux HOWTO peuvent être reproduits et distribués en totalité ou en partie, sur tout support physique ou électronique, tant que cette notice de droit d'auteur est présente sur chaque copie. La redistribution commerciale est autorisée et encouragée ; néammoins, l'auteur souhaite être informé de toute distribution de ce genre. Toute traduction, travail dérivé, ou agrégat incorporant tout ou partie d'un ou plusieurs documents Linux HOWTO doit être couvert par ce même droit d'auteur. Ce qui veut dire que vous ne pouvez produire un travail dérivé d'un HOWTO et imposer des restrictions supplémentaires concernant sa distribution.

En un mot, si votre dorsale STM-64 est tombée ou distribue de la pornographie à vos estimés clients, cela n'est pas de notre faute. Désolé.

Copiez et distribuez (vendez ou donnez) librement ce document, dans n'importe quel format. Les demandes de corrections et/ou de commentaires sont à adresser à la personne qui maintient ce document. Vous pouvez créer un travail dérivé et le distribuer à condition :

1. d'envoyer votre travail dérivé (dans le format le plus approprié, tel que sgml) au LDP (Linux Documentation Project), ou autre, pour une diffusion sur Internet. Si ce n'est pas au LDP, faites savoir au LDP où il est disponible ;
2. que la licence de votre travail dérivé soit la même, ou utilisez la GPL. Incluez une notification du copyright et, au moins, un pointeur vers la licence utilisée ;
3. de créditer pour leur travail les auteurs précédents et les principaux contributeurs.

Si vous envisagez un travail dérivé autre qu'une traduction, vous êtes invité à discuter de vos projets avec le mainteneur actuel.

Il est aussi demandé que, si vous publiez cet HOWTO sur un support papier, vous en envoyiez des exemplaires aux auteurs pour une "relecture critique" :-)

2.2 Connaissances préalables

Comme le titre l'implique, ceci est un HOWTO "avancé". Bien qu'il ne soit pas besoin d'être un expert réseau, certains prérequis sont nécessaires. Ce document se veut la suite du *Linux 2.4 Networking HOWTO* `<http://www.ds9a.nl/2.4Networking/>` par les même auteurs. Vous devriez probablement le lire en premier.

Voici d'autres références qui pourront vous aider à en apprendre plus :

Rusty Russels networking-concepts-HOWTO `<http://netfilter.kernelnotes.org/unreliable-guides/networking-concepts-HOWTO.html>`

Très bonne introduction, expliquant ce qu'est un réseau, et comment on le connecte à d'autres réseaux.

Linux Networking-HOWTO (ex Net-3 HOWTO)

Excellent document, bien que très bavard. Il vous apprendra beaucoup de choses qui sont déjà configurées si vous êtes capable de vous connecter à Internet. Probablement à `/usr/doc/HOWTO/NET-HOWTO.txt` mais peut aussi être trouvé *en ligne* `<http://www.linuxports.com/howto/networking>`

2.3 Ce que Linux peut faire pour vous

Une petite liste des choses qui sont possibles :

- Maîtriser la bande passante pour certains ordinateurs
- Maîtriser la bande passante vers certains ordinateurs
- Vous aider à partager équitablement votre bande passante
- Protéger votre réseau des attaques de type Déni de Service
- Protéger le réseau local des accès de vos clients
- Multiplexer plusieurs serveurs en un seul, pour l'équilibrage de charge, ou une disponibilité améliorée
- Restreindre l'accès à vos ordinateurs
- Limiter l'accès de vos utilisateurs vers d'autres hôtes
- Faire du routage basé sur l'ID utilisateur (eh oui !), l'adresse MAC, l'adresse IP source, le port, le type de service, l'heure, ou le contenu.

Peu de personnes utilisent couramment ces fonctionnalités avancées. Il y a plusieurs raisons à cela. Bien que la documentation soit fournie, la prise en main est difficile. Les commandes de contrôle du trafic ne sont pratiquement pas documentées.

2.4 Notes diverses

Il y a plusieurs choses qui doivent être notées au sujet de ce document. Bien que j'en aie écrit la majeure partie, je ne veux vraiment pas qu'il reste tel quel. Je crois beaucoup à l'Open Source, je vous encourage donc à envoyer des remarques, des mises à jour, des corrections, etc. N'hésitez pas à m'avertir des coquilles ou d'erreurs pures et simples. Si mon anglais vous paraît parfois peu naturel, ayez en tête, s'il vous plaît, que l'anglais n'est pas ma langue natale. N'hésitez pas à m'envoyer vos suggestions [NdT : en anglais !]

Si vous pensez que vous êtes plus qualifié que moi pour maintenir une section, ou si vous pensez que vous pouvez écrire et maintenir de nouvelles sections, vous êtes le bienvenu. Le source SGML de cette documentation est disponible via CVS, ce qui permet à plus de personnes de travailler dessus.

Pour vous aider, vous trouverez beaucoup de mentions FIXME (NdT : "à corriger"). Les corrections sont toujours les bienvenues. Si vous trouvez une mention FIXME, vous saurez que vous êtes en territoire inconnu. Cela ne veut pas dire qu'il n'y a pas d'erreurs ailleurs, faites donc très attention. Si vous avez validé quelque chose, faites-le nous savoir, ce qui nous permettra de retirer la mention FIXME.

Je prendrai quelques libertés tout au long de cet HOWTO. Par exemple, je pars de l'hypothèse d'une connexion Internet à 10 Mbits, bien que je sache très bien que cela n'est pas vraiment courant.

2.5 Accès, CVS et propositions de mises à jour

L'adresse canonique de cet HOWTO est *Ici* `<http://www.ds9a.nl/2.4Routing>`.

Nous avons maintenant un CVS en accès anonyme disponible depuis le monde entier. Cela est intéressant pour plusieurs raisons. Vous pouvez facilement télécharger les nouvelles versions de ce HOWTO et soumettre des patches. En outre, cela permet aux auteurs de travailler sur le source de facon indépendante, ce qui est une bonne chose aussi.

```
$ export CVSROOT=:pserver:anon@outpost.ds9a.nl:/var/cvsroot
$ cvs login
CVS password: [enter 'cvs' (without 's)]
$ cvs co 2.4routing
cvs server: Updating 2.4routing
U 2.4routing/2.4routing.sgml
```

Si vous repérez une erreur, ou voulez ajouter quelque chose, faites le en local, exécutez `cvs diff -u`, et envoyez-nous le résultat.

Un fichier Makefile est fourni pour vous aider à créer des fichiers PostScript, dvi, pdf, html et texte. Vous pouvez avoir à installer les sgml-tools, ghostscript et tetex pour obtenir tous les formats de sortie.

2.6 Plan du document

Nous allons essayer de faire des manipulations intéressantes dès le début, ce qui veut dire que tout ne sera pas expliqué en détail tout de suite. Veuillez passer sur ces détails, et accepter de considérer qu'ils deviendront clairs par la suite.

Le routage et le filtrage sont deux choses distinctes. Le filtrage est très bien documenté dans le HOWTO de Rusty, disponible ici :

– *Rusty's Remarkably Unreliable Guides* `<http://netfilter.kernelnotes.org/unreliable-guides/>`

Nous nous focaliserons principalement sur ce qu'il est possible de faire en combinant netfilter et iproute2.

3 Introduction à iproute2

3.1 Pourquoi iproute2 ?

La plupart des distributions Linux et des UNIX utilisent couramment les vénérables commandes "arp", "ifconfig" et "route". Bien que ces outils fonctionnent, ils montrent quelques comportements inattendus avec les noyaux Linux des séries 2.2 et plus. Par exemple, les tunnels GRE font partie intégrante du routage de nos jours, mais ils nécessitent des outils complètement différents.

Avec iproute2, les tunnels font partie intégrante des outils.

Les noyaux Linux des séries 2.2 et plus ont un sous-système réseau complètement réécrit. Ce nouveau codage de la partie réseau apporte à Linux des performances et des fonctionnalités qui n'ont pratiquement pas d'équivalent dans les autres systèmes d'exploitation. En fait, le nouveau logiciel de filtrage, routage et de classification possède plus de fonctionnalités que les logiciels fournis sur beaucoup de routeurs dédiés, de pare-feux et de produits de mise en forme (shaping) du trafic.

Dans les systèmes d'exploitation existants, au fur et à mesure que de nouveaux concepts réseau apparaissaient, les développeurs sont parvenus à les greffer sur les structures existantes. Ce travail constant d'empilage de couches a conduit à des codes réseau aux comportements étranges, un peu comme les langues humaines. Dans le passé, Linux émulait le mode de fonctionnement de SunOS, ce qui n'était pas l'idéal.

La nouvelle structure d'iproute2 a permis de formuler clairement des fonctionnalités impossibles à implanter dans le sous-système réseau précédent.

3.2 Un tour d'horizon d'Iproute2

Linux possède un système sophistiqué d'allocation de bande passante appelé Contrôle de trafic (Traffic Control). Ce système supporte différentes méthodes pour classer, ranger par ordre de priorité, partager et limiter le trafic entrant et sortant.

Nous commencerons par un petit tour d'horizon des possibilités d'iproute2.

3.3 Prérequis

Vous devez être sûr que vous avez installé les outils utilisateur (NdT : userland tools, par opposition à la partie "noyau" d'iproute2). Le paquet concerné s'appelle "iproute" sur RedHat et Debian. Autrement, il peut être trouvé à `ftp ://ftp.inr.ac.ru/ip-routing/iproute2-2.2.4-now-ss??????.tar.gz`. Certains éléments d'iproute vous imposent l'activation de certaines options du noyau.

FIXME : Nous devrions mentionner *iproute2-current.tar.gz* `<ftp://ftp.inr.ac.ru/ip-routing/iproute2-current.tar.gz>` qui est toujours la dernière version.

3.4 Explorer votre configuration courante

Cela peut vous paraître surprenant, mais iproute2 est déjà configuré ! Les commandes courantes `ifconfig` et `route` utilisent déjà les appels système avancés d'iproute2, mais essentiellement avec les options par défaut (c'est-à-dire ennuyeuses).

L'outil `ip` est central, et nous allons lui demander de nous montrer les interfaces.

3.4.1 `ip` nous montre nos liens

```
[ahu@home ahu]$ ip link list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: dummy: <BROADCAST,NOARP> mtu 1500 qdisc noop
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1400 qdisc pfifo_fast qlen 100
    link/ether 48:54:e8:2a:47:16 brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:e0:4c:39:24:78 brd ff:ff:ff:ff:ff:ff
3764: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 10
    link/ppp
```

La sortie peut varier, mais voici ce qui est affiché pour mon routeur NAT (NdT : translation d'adresse) chez moi. J'expliquerai seulement une partie de la sortie, dans la mesure où tout n'est pas directement pertinent.

La première interface que nous voyons est l'interface loopback. Bien que votre ordinateur puisse fonctionner sans, je vous le déconseille. La taille de MTU (unité maximum de transmission) est de 3924 octets, et loopback n'est pas supposé être mis en file d'attente, ce qui prend tout son sens dans la mesure où cette interface est le fruit de l'imagination de votre noyau.

Je vais passer sur l'interface dummy pour l'instant, et il se peut qu'elle ne soit pas présente sur votre ordinateur. Il y a ensuite mes deux interfaces réseau, l'une du côté de mon modem câble, l'autre servant mon segment ethernet à la maison. De plus, nous voyons une interface ppp0.

Notons l'absence d'adresses IP. Iproute déconnecte les concepts de "liens" et "d'adresses IP". Avec l'IP aliasing, le concept de l'adresse IP canonique est devenu, de toute façon, sans signification.

`ip` nous montre bien, cependant, l'adresse MAC, l'identifiant matériel de nos interfaces ethernet.

3.4.2 `ip` nous montre nos adresses IP

```
[ahu@home ahu]$ ip address show
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: dummy: <BROADCAST,NOARP> mtu 1500 qdisc noop
```

```
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1400 qdisc pfifo_fast qlen 100
    link/ether 48:54:e8:2a:47:16 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/8 brd 10.255.255.255 scope global eth0
4: eth1: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:e0:4c:39:24:78 brd ff:ff:ff:ff:ff:ff
3764: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 10
    link/ppp
    inet 212.64.94.251 peer 212.64.94.1/32 scope global ppp0
```

Cela contient plus d'informations : ip montre toutes nos adresses, et à quelles cartes elles appartiennent. "inet" désigne Internet. Il y a beaucoup d'autres familles d'adresses, mais elles ne nous concernent pas pour le moment.

Examinons eth0 de plus près. Il est dit qu'il est relié à l'adresse internet "10.0.0.1/8". Qu'est-ce que cela signifie ? Le /8 désigne le nombre de bits réservés à l'adresse réseau. Il y a 32 bits, donc il reste 24 bits pour désigner une partie de notre réseau. Les 8 premiers bits de 10.0.0.1 correspondent à 10.0.0.0, notre adresse réseau, et notre masque de sous-réseau est 255.0.0.0.

Les autres bits repèrent des machines directement connectées à cette interface, donc 10.250.3.13 est directement disponible sur eth0, comme l'est 10.0.0.1 dans notre exemple.

Avec ppp0, le même concept existe, bien que les nombres soient différents. Son adresse est 212.64.94.251, sans masque de sous-réseau. Cela signifie que vous avez une liaison point à point et que toutes les adresses, à l'exception de 212.64.94.251, sont distantes. Il y a cependant plus d'informations. En effet, on nous dit que de l'autre côté du lien, il n'y a encore qu'une seule adresse, 212.64.94.1. Le /32 nous précise qu'il n'y a pas de "bits réseau".

Il est absolument vital que vous compreniez ces concepts. Référez-vous à la documentation mentionnée au début de cet HOWTO si vous avez des doutes.

Vous pouvez aussi noter "qdisc", qui désigne la "Queueing Discipline" (gestion de la mise en file d'attente). Cela deviendra vital plus tard.

3.4.3 `ip` nous montre nos routes

Nous savons maintenant comment trouver les adresses 10.x.y.z, et nous sommes capables d'atteindre 212.64.94.1. Cela n'est cependant pas suffisant, et nous avons besoin d'instructions pour atteindre le monde. L'Internet est disponible via notre connexion ppp, et il se trouve que 212.64.94.1 est prêt à propager nos paquets à travers le monde, et à nous renvoyer le résultat.

```
[ahu@home ahu]$ ip route show
212.64.94.1 dev ppp0  proto kernel  scope link  src 212.64.94.251
10.0.0.0/8 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 212.64.94.1 dev ppp0
```

Cela se comprend de soi-même. Les 4 premières lignes donnent explicitement ce qui était sous-entendu par `ip address show`, la dernière ligne nous indiquant que le reste du monde peut être trouvé via 212.64.94.1, notre passerelle par défaut. Nous pouvous voir que c'est une passerelle à cause du mot "via", qui nous indique que nous avons besoin d'envoyer les paquets vers 212.64.94.1, et que c'est elle qui se chargera de tout.

En référence, voici ce que l'ancien utilitaire "route" nous propose :

```
[ahu@home ahu]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
```

```
Iface
212.64.94.1     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         212.64.94.1     0.0.0.0         UG    0      0        0 ppp0
```

3.5 ARP

ARP est le Protocole de Résolution d'Adresse (Address Resolution Protocol). Il est décrit dans le *RFC 826* `<http://www.faqs.org/rfcs/rfc826.html>`. ARP est utilisé par une machine d'un réseau local pour retrouver l'adresse matérielle (la localisation) d'une autre machine sur le même réseau. Les machines sur Internet sont généralement connues par leur nom auquel correspond une adresse IP. C'est ainsi qu'une machine sur le réseau foo.com est capable de communiquer avec une autre machine qui est sur le réseau bar.net. Une adresse IP, cependant, ne peut pas vous indiquer la localisation physique de la machine. C'est ici que le protocole ARP entre en jeu.

Prenons un exemple très simple. Supposons que j'aie un réseau composé de plusieurs machines, dont la machine "foo" d'adresse IP 10.0.0.1 et la machine "bar" qui a l'adresse IP 10.0.0.2. Maintenant, foo veut envoyer un "ping" vers bar pour voir s'il est actif, mais foo n'a aucune indication sur la localisation de bar. Donc, si foo décide d'envoyer un "ping" vers bar, il a besoin d'envoyer une requête ARP. Cette requête ARP est une façon pour foo de crier sur le réseau "Bar (10.0.0.2) ! Où es-tu ?". Par conséquent, toutes les machines sur le réseau entendront foo crier, mais seul bar (10.0.0.2) répondra. Bar enverra une réponse ARP directement à foo ; ce qui revient à dire : "Foo (10.0.0.1) ! je suis ici, à l'adresse 00 :60 :94 :E :08 :12". Après cette simple transaction utilisée pour localiser son ami sur le réseau, foo est capable de communiquer avec bar jusqu'à ce qu'il (le cache ARP de foo) oublie où bar est situé.

Maintenant, voyons comment cela fonctionne. Vous pouvez consulter votre cache (table) arp (neighbor) comme ceci :

```
[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud reachable
```

Comme vous pouvez le voir, ma machine espa041 (9.3.76.41) sait où trouver espa042 (9.3.76.42) et espagate (9.3.76.1). Maintenant, ajoutons une autre machine dans le cache arp.

```
[root@espa041 /home/paulsch/.gnome-desktop]# ping -c 1 espa043
PING espa043.austin.ibm.com (9.3.76.43) from 9.3.76.41 : 56(84) bytes of data.
64 bytes from 9.3.76.43: icmp_seq=0 ttl=255 time=0.9 ms

--- espa043.austin.ibm.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.9/0.9/0.9 ms

[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.43 dev eth0 lladdr 00:06:29:21:80:20 nud reachable
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud reachable
```

Par conséquent, lorsque espa041 a essayé de contacter espa043, l'adresse matérielle de espa043 (sa localisation) a alors été ajoutée dans le cache ARP. Donc, tant que la durée de vie de l'entrée correspondant à espa043 dans le cache ARP n'est pas dépassée, espa041 sait localiser espa043 et n'a plus besoin d'envoyer de requête ARP.

Maintenant, effaçons espa043 de notre cache ARP.

```
[root@espa041 /home/src/iputils]# ip neigh delete 9.3.76.43 dev eth0
[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.43 dev eth0  nud failed
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud stale
```

Maintenant, espa041 a à nouveau oublié la localisation d'espa043 et aura besoin d'envoyer une autre requête ARP la prochaine fois qu'il voudra communiquer avec lui. Vous pouvez aussi voir ci-dessus que l'état d'espagate (9.3.76.1) est passé en "stale". Cela signifie que la localisation connue est encore valide, mais qu'elle devra être confirmée à la première transaction avec cette machine.

4 Règles - bases de données des politiques de routage

Si vous avez un routeur important, il se peut que vous vouliez satisfaire les besoins de différentes personnes, qui peuvent être traitées différemment. Les bases de données des politiques de routage vous aident à faire cela, en gérant plusieurs ensembles de tables de routage. Si vous voulez utiliser cette fonctionnalité, assurez vous que le noyau est compilé avec l'option "IP : policy routing".

Quand le noyau doit prendre une décision de routage, il recherche quelle table consulter. Par défaut, il y a trois tables. L'ancien outil `route` modifie les tables "main" (principale) et "local" (locale), comme le fait l'outil `ip` (par défaut).

Les règles par défaut :

```
[ahu@home ahu]$ ip rule list
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

Cela liste les règles, accompagnées de leurs priorités. Nous voyons que toutes les règles sont appliquées à tous les paquets ("from all"). Nous avons vu la table "main" précédemment, sa sortie s'effectuant avec `ip route ls`, mais les tables "local" et "default" sont nouvelles.

Si nous voulons faire des choses fantaisistes, nous pouvons créer des règles qui pointent vers des tables différentes et qui nous permettent de redéfinir les règles de routage du système.

Pour savoir exactement ce que fait le noyau en présence d'un assortiment de règles plus complet, référez-vous à la documentation ip-cref d'Alexey [NdT : dans le paquet iproute de votre distribution].

4.1 Routage simple par l'adresse source

Prenons encore une fois un exemple réel. J'ai 2 modems câble, connectés à un routeur Linux NAT ("masquerading"). Les personnes habitant avec moi me paient pour avoir accès à Internet. Supposons qu'un de mes co-locataires visite seulement hotmail et veuille payer moins. C'est d'accord pour moi, mais il utilisera le modem le plus lent.

Le modem câble "rapide" est connu sous 212.64.94.251 et est en liaison PPP avec 212.64.94.1. Le modem câble "lent" est connu sous diverses adresses ip, 212.64.78.148 dans notre exemple avec un lien vers 195.96.98.253.

La table locale :

```
[ahu@home ahu]$ ip route list table local
broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
local 10.0.0.1 dev eth0  proto kernel  scope host  src 10.0.0.1
broadcast 10.0.0.0 dev eth0  proto kernel  scope link  src 10.0.0.1
local 212.64.94.251 dev ppp0  proto kernel  scope host  src 212.64.94.251
```

```
broadcast 10.255.255.255 dev eth0  proto kernel  scope link  src 10.0.0.1
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 212.64.78.148 dev ppp2  proto kernel  scope host  src 212.64.78.148
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
```

Il y a beaucoup de choses évidentes, mais aussi des choses qui ont besoin d'être précisées quelque peu, ce que nous allons faire. La table de routage par défaut est vide.

Regardons la table principale ("main") :

```
[ahu@home ahu]$ ip route list table main
195.96.98.253 dev ppp2  proto kernel  scope link  src 212.64.78.148
212.64.94.1 dev ppp0  proto kernel  scope link  src 212.64.94.251
10.0.0.0/8 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 212.64.94.1 dev ppp0
```

Maintenant, nous générons une nouvelle règle que nous appellerons "John", pour notre hypothétique colocataire. Bien que nous puissions travailler avec des nombres IP purs, il est plus facile d'ajouter notre table dans le fichier `/etc/iproute2/rt\_tables`.

```
# echo 200 John >> /etc/iproute2/rt_tables
# ip rule add from 10.0.0.10 table John
# ip rule ls
0:      from all lookup local
32765:  from 10.0.0.10 lookup John
32766:  from all lookup main
32767:  from all lookup default
```

Maintenant, tout ce qu'il reste à faire est de générer la table "John", et de vider le cache des routes :

```
# ip route add default via 195.96.98.253 dev ppp2 table John
# ip route flush cache
```

Et voilà qui est fait. Il ne reste plus, comme exercice laissé au lecteur, qu'à implanter cela dans ip-up.

5 GRE et autres tunnels

Il y a trois sortes de tunnels sous Linux : l'IP dans un tunnel IP, le tunnel GRE et les tunnels qui existent en dehors du noyau (comme, par exemple, PPTP).

5.1 Quelques remarques générales à propos des tunnels :

Les tunnels peuvent faire des choses très inhabituelles et vraiment sympa. Ils peuvent aussi absolument tout détraquer si vous ne les avez pas configurés correctement. Ne définissez pas votre route par défaut sur un tunnel, à moins que vous ne sachiez **exactement** ce que vous faites.

De plus, le passage par un tunnel augmente le poids des en-têtes (overhead), puisqu'un en-tête IP supplémentaire est nécessaire. Typiquement, ce surcoût est de 20 octets par paquet. Donc, si la taille maximum de votre paquet sur votre réseau (MTU) est de 1500 octets, un paquet qui est envoyé à travers un tunnel sera limité à une taille de 1480 octets. Cela n'est pas nécessairement un problème, mais soyez sûr d'avoir bien étudié la fragmentation et le réassemblage des paquets IP quand vous prévoyez de relier des réseaux de grande taille par des tunnels. Et bien sûr, la manière la plus rapide de creuser un tunnel est de creuser des deux côtés.

5.2 IP dans un tunnel IP

Ce type de tunnel est disponible dans Linux depuis un long moment. Il nécessite deux modules, ipip.o et new_tunnel.o.

Disons que vous avez trois réseaux : 2 réseaux internes A et B, et un réseau intermédiaire C (ou disons Internet). Les caractéristiques du réseau A sont :

```
réseau 10.0.1.0
masque de sous-réseau 255.255.255.0
routeur  10.0.1.1
```

Le routeur a l'adresse 172.16.17.18 sur le réseau C.

et le réseau B :

```
réseau 10.0.2.0
masque de sous-réseau 255.255.255.0
routeur  10.0.2.1
```

Le routeur a l'adresse 172.19.20.21 sur le réseau C.

En ce qui concerne le réseau C, nous supposerons qu'il transmettra n'importe quel paquet de A vers B et vice-versa. Il est également possible d'utiliser l'Internet pour cela.

Voici ce qu'il faut faire :

Premièrement, assurez-vous que les modules sont installés :

```
insmod ipip.o
insmod new_tunnel.o
```

Ensuite, sur le routeur du réseau A, faites la chose suivante :

```
ifconfig tunl0 10.0.1.1 pointopoint 172.19.20.21
route add -net 10.0.2.0 netmask 255.255.255.0 dev tunl0
```

et sur le routeur du réseau B :

```
ifconfig tunl0 10.0.2.1 pointopoint 172.16.17.18
route add -net 10.0.1.0 netmask 255.255.255.0 dev tunl0
```

Et quand vous aurez terminé avec votre tunnel :

```
ifconfig tunl0 down
```

Vite fait, bien fait. Vous ne pouvez pas transmettre les paquets de diffusion (broadcast), ni le trafic IPv6 à travers un tunnel IP-IP. Vous ne pouvez connecter que deux réseaux IPv4 qui, normalement, ne seraient pas capables de se "parler", c'est tout. Dans la mesure où la compatibilité a été conservée, ce code tourne depuis un bon moment, et il reste compatible depuis les noyaux 1.3. Le tunnel Linux IP dans IP ne fonctionne pas avec d'autres systèmes d'exploitation ou routeurs, pour autant que je sache. C'est simple, ça marche. Utilisez-le si vous le pouvez, autrement utilisez GRE.

5.3 Le tunnel GRE

GRE est un protocole de tunnel qui a été à l'origine développé par Cisco, et qui peut réaliser plus de choses que le tunnel IP dans IP. Par exemple, vous pouvez aussi transporter du trafic multi-diffusion (multicast) et de l'IPv6 à travers un tunnel GRE.

Dans Linux, vous aurez besoin du module ip_gre.

5.3.1 Le tunnel IPv4

Dans un premier temps, intéressons-nous au tunnel IPv4 :

Disons que vous avez trois réseaux : 2 réseaux internes A et B, et un réseau intermédiaire C (ou disons Internet).

Les caractéristiques du réseau A sont :

```
réseau 10.0.1.0
masque de sous-réseau 255.255.255.0
routeur  10.0.1.1
```

Le routeur a l'adresse 172.16.17.18 sur le réseau C. Appelons ce réseau neta.

Et pour le réseau B :

```
réseau 10.0.2.0
masque de sous-réseau 255.255.255.0
routeur  10.0.2.1
```

Le routeur a l'adresse 172.19.20.21 sur le réseau C. Appelons ce réseau netb.

En ce qui concerne le réseau C, nous supposerons qu'il transmettra n'importe quels paquets de A vers B et vice-versa. Comment et pourquoi, on s'en fiche.

Sur le routeur du réseau A, nous faisons la chose suivante :

```
ip tunnel add netb mode gre remote 172.19.20.21 local 172.16.17.18 ttl 255
ip addr add 10.0.1.1 dev netb
ip route add 10.0.2.0/24 dev netb
```

Discutons un peu de cela. Sur la ligne 1, nous avons ajouté un périphérique tunnel, que nous avons appelé netb (ce qui est évident, dans la mesure où c'est là que nous voulons aller). De plus, nous lui avons dit d'utiliser le protocole GRE (mode gre), que l'adresse distante est 172.19.20.21 (le routeur de l'autre coté), que nos paquets "tunnelés" devront être générés à partir de 172.16.17.18 (ce qui autorise votre serveur à avoir plusieurs adresses IP sur le réseau C et ainsi vous permet de choisir laquelle sera utilisée pour votre tunnel) et que le champ TTL de vos paquets sera fixé à 255 (ttl 255).

Sur la deuxième ligne, nous avons donné à cette nouvelle interface l'adresse 10.0.1.1. C'est bon pour de petits réseaux, mais quand vous commencez une exploitation minière (BEAUCOUP de tunnels !), vous pouvez utiliser une autre gamme d'adresses IP pour vos interfaces "tunnel" (dans cet exemple, vous pourriez utiliser 10.0.3.0).

Sur la troisième ligne, nous positionnons une route pour le réseau B. Notez la notation différente pour le masque de sous-réseau. Si vous n'êtes pas familiarisé avec cette notation, voici comment ça marche : vous écrivez le masque de sous-réseau sous sa forme binaire, et vous comptez tous les 1. Si vous ne savez pas comment faire cela, rappelez-vous juste que 255.0.0.0 est /8, 255.255.0.0 est /16 et 255.255.255.0 est /24. Et 255.255.254.0 est /23, au cas où ça vous intéresserait.

Mais arrêtons ici, et continuons avec le routeur du réseau B.

```
ip tunnel add neta mode gre remote 172.16.17.18 local 172.19.20.21 ttl 255
ip addr add 10.0.2.1 dev neta
ip route add 10.0.1.0/24 dev neta
```

Et quand vous voudrez retirer le tunnel sur le routeur A :

```
ip link set netb down
ip tunnel del netb
```

Bien sûr, vous pouvez remplacer netb par neta pour le routeur B.

5.3.2 Le tunnel IPv6

BON GROS AVERTISSEMENT !

Ce qui suit n'est pas testé. Vous opérez à vos risques et périls. Ne dites pas que je ne vous avais pas prévenu.

FIXME : vérifier et essayer tout ceci.

De petites choses à propos des adresses IPv6 :

Les adresses IPv6 sont, en comparaison avec les adresses IPv4, monstrueusement grosses. Voici un exemple :

`3ffe:2502:200:40:281:48fe:dcfe:d9bc`

Donc, pour rendre l'écriture plus facile, il y a quelques règles :
- Ne pas utiliser les zéros de tête, comme dans IPv4 ;
- Utiliser des double-points de séparation tous les 16 bits (2 octets) ;
- Quand vous avez beaucoup de zéros consécutifs, vous pouvez écrire `::`. Vous ne pouvez faire cela qu'une seule fois par adresse et seulement pour une longueur de 16 bits.

En utilisant ces règles, l'addresse 3ffe :0000 :0000 :0000 :0000 :0020 :34A1 :F32C peut être écrite 3ffe : :20 :34A1 :F32C, ce qui est beaucoup plus court.

À propos des tunnels.

Supposons que vous ayez le réseau IPv6 suivant, et que vous vouliez le connecter à une dorsale IPv6 (6bone), ou à un ami.

```
Réseau 3ffe:406:5:1:5:a:2:1/96
```

Votre adresse IPv4 est 172.16.17.18, et le routeur 6bone a une adresse IPv4 172.22.23.24.

```
ip tunnel add sixbone mode sit remote 172.22.23.24 local 172.16.17.18 ttl 255
ip link set sixbone up
ip addr add 3ffe:406:5:1:5:a:2:1/96 dev sixbone
ip route add 3ffe::/15 dev sixbone
```

Voyons cela de plus près. Sur la première ligne, nous avons créé un périphérique tunnel appelé sixbone. Nous lui avons affecté le mode "sit" (qui est le tunnel IPv6 sur IPv4) et lui avons dit où l'on va (remote) et d'où l'on vient (local). TTL est configuré à son maximum, 255. Ensuite, nous avons rendu le périphérique actif (up). Puis, nous avons ajouté notre propre adresse réseau et configuré une route pour 3ffe : :/15 à travers le tunnel.

Les tunnels GRE constituent actuellement le type préféré de tunnel. C'est un standard qui est largement adopté, même à l'extérieur de la communauté Linux, ce qui constitue une bonne raison de l'utiliser.

5.4 Tunnels dans l'espace utilisateur

Il y a des dizaines d'implémentations de tunnels à l'extérieur du noyau. Les plus connus sont bien sûr PPP et PPTP, mais il y en a bien plus (certains propriétaires, certains sécurisés, d'autres qui n'utilisent pas IP), qui dépassent le cadre de cet HOWTO.

6 IPsec : IP sécurisé à travers l'Internet

FIXME : Attendre notre rédacteur vedette Stefan pour finir cette partie.

7 Routage multi-diffusion (multicast)

FIXME : Pas de rédacteur !

8 Utilisation de la mise en file d'attente basée sur des classes ("Class Based Queueing") pour la gestion de la bande passante

Quand je l'ai découvert, cela m'a vraiment "soufflé". Linux 2.2 contient toutes les fonctionnalités pour la gestion de la bande passante, de manière comparable à un système dédié de haut niveau.

Linux dépasse même ce que l'ATM et le Frame peuvent fournir.

Les deux unités de base du Contrôle de Trafic sont les filtres et les files d'attente. Les filtres placent le trafic dans des files d'attente, qui recueillent ainsi le trafic et décident ce qu'il faut envoyer en premier, envoyer plus tard, ou éliminer. Il existe plusieurs variantes de filtres et de files d'attente.

Les filtres les plus communs sont fwmark et u32. Le premier vous permet d'utiliser le code Netfilter de Linux pour sélectionner le trafic, et le second vous permet de sélectionner ce trafic à partir de N'IMPORTE QUEL en-tête. La file d'attente la plus connue est la Class Based Queue (File d'attente basée sur des classes). CBQ est une super-file d'attente, qui peut contenir d'autres files d'attente (même d'autres CBQ).

Il n'est peut-être pas évident de voir ce que la mise en file d'attente peut avoir à faire avec la gestion de bande passante, mais ça marche vraiment.

Pour cadre de référence de cette section, j'ai pris l'exemple d'un fournisseur d'accès chez qui j'ai appris les ficelles du métier, pour ne pas le citer : Casema Internet en Hollande. Casema, qui est actuellement un câble-opérateur, a des besoins internet pour ses clients et pour son propre compte. La plupart des ordinateurs de la société ont un accès Internet. En réalité, ils ont beaucoup d'argent à dépenser et n'utilisent pas Linux pour la gestion de la bande passante.

Nous étudierons comment notre fournisseur d'accès aurait pu utiliser Linux pour gérer sa bande passante.

8.1 Qu'est-ce que la mise en file d'attente ?

Avec la mise en file d'attente, nous déterminons l'ordre dans lequel les données sont envoyées. Il est important de le comprendre, nous ne pouvons que mettre en forme les données que nous transmettons. Comment ce changement de prioriré détermine-t-il la vitesse de transmission ?

Imaginez une caisse enregistreuse capable de traiter 3 clients par minute. Les personnes souhaitant payer vont attendre en file indienne en bout de queue. C'est une "file d'attente fifo". (NdT : fifo = First In, First Out, premier entré, premier sorti) Supposons maintenant que vous laissiez toujours certaines personnes s'insérer au milieu de la queue, et non en fin. Ces personnes attendront moins de temps et donc pourront faire leurs courses plus rapidement.

Avec la manière dont Internet travaille, nous n'avons pas de contrôle direct de ce que les personnes nous envoient. C'est un peu comme votre boîte aux lettres (physique !) chez vous. Il n'y a pas de façon d'influencer le nombre de lettres que vous recevez, à moins de contacter tout le monde.

Cependant, l'Internet est principalement basé sur TCP/IP qui possède quelques fonctionnalités qui vont pouvoir nous aider. TCP/IP n'a pas d'aptitude à connaître les performances d'un réseau entre deux hôtes. Il commence à envoyer des paquets, de plus en plus rapidement ('slow start') et quand des paquets commencent à se perdre, il ralentit.

C'est comme si vous ne lisiez que la moitié de votre courrier en espérant que vos correspondants arrêteront de vous en envoyer. À la différence du courrier, ça marche sur l'Internet :-)

FIXME : expliquer comment, normalement, les acquittements (ACKs) sont utilisés pour déterminer la vitesse.

```
[L'Internet] ---<E3, T3, peu importe>--- [Routeur Linux] --- [Bureau+FAI]
                                      eth1          eth0
```

Maintenant, notre routeur Linux a deux interfaces que je vais appeler eth0 et eth1. Eth1 est connecté à notre routeur qui transmet les paquets venant de ou allant vers notre fibre optique.

Eth0 est connecté à un sous-réseau qui contient à la fois le pare-feu de la société et notre équipement de tête, à travers lequel nos clients peuvent se connecter.

Dans la mesure où nous ne pouvons limiter que ce que nous envoyons, nous avons besoin de deux ensembles de règles séparés, mais très similaires. En modifiant la mise en file d'attente sur eth0, nous déterminons à quelle vitesse les données reçues sont envoyées à nos clients. On détermine ainsi la bande passante descendante disponible vers eux, en résumé, leur "vitesse de téléchargement".

Sur eth1, nous déterminons à quelle vitesse nous envoyons les données sur Internet, à quelle vitesse nos utilisateurs, à la fois ceux de la société et les clients, peuvent émettre les données.

8.2 Première tentative de partage de bande passante

CBQ nous permet de créer plusieurs classes, et même des classes incluses dans d'autres classes. Les divisions les plus larges peuvent être appelées des agences. Au sein de ces classes, se trouveraient des classes comme "traitement par lot" et "traitement interactif".

Par exemple, nous pourrions avoir une connexion Internet de 10 Mbits devant être partagée par nos clients et les besoins de la société. Nous ne devons pas permettre à quelques personnes de la société de monopoliser un gros volume de bande passante, qui devrait être vendue à nos clients.

D'un autre côté, nos clients ne doivent pas pouvoir grignoter le trafic réservé à nos bureaux.

Avant, une manière de résoudre cela était d'utiliser le Frame relay/ATM et de créer des circuits virtuels. Cela fonctionne, mais les réseaux Frame Relay ne sont pas suffisament maillés et ATM est terriblement inefficace dans le transport du trafic IP. Ni l'un ni l'autre ne possède de moyens standardisés pour distribuer le trafic sur différents circuits virtuels (VC).

Cependant, si vous utilisez ATM, Linux peut habilement réaliser des tâches de classification de trafic pour vous. Une autre manière de procéder consiste à commander plusieurs connexions séparées mais ce n'est ni pratique ni élégant et ça ne résoudra pas tous vos problèmes.

CBQ à la rescousse !

Clairement, nous avons ici deux classes principales, "ISP" et "Office". Au départ, on ne se préoccupe pas vraiment de ce que les divisions font de leur bande passante, et donc, nous ne subdiviserons pas plus loin leurs classes.

Nous décidons que les clients devront toujours avoir 8 Mbits de trafic descendant garanti, et nos bureaux 2 Mbits.

La configuration du contrôle de trafic est faite avec l'outil `tc`, du paquet iproute2.

```
# tc qdisc add dev eth0 root handle 10: cbq bandwidth 10Mbit avpkt 1000
```

Bon, beaucoup de chiffres ici. Qu'avons-nous fait ? Nous avons configuré la gestion de la mise en file d'attente (queuing discipline) de eth0. Par "root", nous signifions que cette file d'attente est la racine. Nous lui avons donné la référence (handle) "10 :". Nous voulons faire du "CBQ", nous le mentionnons donc sur la ligne de commande. Nous indiquons au noyau qu'il peut allouer 10 Mbits et que la taille moyenne d'un paquet est aux alentours de 1000 octets.

FIXME : Re-vérifier avec Alexey que le calcul de la cellule intégrée est suffisant.

FIXME : Avec un MTU de 1500, la cellule par défaut est calculée de la même façon que dans l'ancien exemple.

FIXME : J'ai vérifié les sources (espaces utilisateur et noyau), on doit pouvoir l'omettre sans problème.

Maintenant, nous devons générer notre classe racine, à partir de laquelle toutes les autres descendront :

```
# tc class add dev eth0 parent 10:0 classid 10:1 cbq bandwidth 10Mbit rate \
  10Mbit allot 1514 weight 1Mbit prio 8 maxburst 20 avpkt 1000
```

Encore des nombres à gérer - l'implémentation CBQ dans Linux est très générique. Avec "parent 10 :0", nous indiquons que cette classe descend de la file d'attente racine générée plus tôt, et référencée par "10 :". Avec "classid 10 :1", nous baptisons cette nouvelle classe.

Nous ne disons vraiment rien de plus au noyau. Nous générons simplement une classe qui englobe toute la bande passante disponible sur l'interface. Nous avons aussi spécifié que le MTU (plus l'en-tête) est de 1514 octets. Nous appliquons une pondération à cette classe avec un paramètre de réglage de 1 Mbit.

Maintenant, nous créons notre classe ISP :

```
# tc class add dev eth0 parent 10:1 classid 10:100 cbq bandwidth 10Mbit rate \
  8Mbit allot 1514 weight 800Kbit prio 5 maxburst 20 avpkt 1000 \
  bounded
```

Nous allouons 8 Mbits et nous indiquons que cette classe ne doit pas dépasser cette limite par l'ajout du paramètre "bounded". Autrement, elle aurait commencé à emprunter de la bande passante aux autres classes. De plus, cette classe pourra prêter sa bande passante à d'autres classes. Nous en discuterons plus tard.

Pour finir, nous générons la classe "Office" (bureau) :

```
# tc class add dev eth0 parent 10:1 classid 10:200 cbq bandwidth 10Mbit rate \
  2Mbit allot 1514 weight 200Kbit prio 5 maxburst 20 avpkt 1000 \
  bounded
```

Pour rendre les choses un peu plus claires, voici un diagramme qui montre nos classes :

```
+-------------[10: 10Mbit]----------------------+
|+-------------[10:1 root 10Mbit]--------------+|
||                                             ||
|| +-[10:100 8Mbit]-+ +--[10:200 2Mbit]-----+  ||
|| |                | |                     |  ||
|| | ISP            | |  Office             |  ||
|| |                | |                     |  ||
|| +----------------+ +---------------------+  ||
||                                             ||
|+---------------------------------------------+|
+-----------------------------------------------+
```

Nous avons maintenant indiqué au noyau quelles sont nos classes, mais pas encore comment gérer les files d'attente. Nous le faisons à présent, d'un seul coup pour les deux classes.

```
# tc qdisc add dev eth0 parent 10:100 sfq quantum 1514b perturb 15
# tc qdisc add dev eth0 parent 10:200 sfq quantum 1514b perturb 15
```

Dans ce cas, nous installons le gestionnaire de mise en file d'attente Stochastic Fairness Queueing (approx. "statistiquement équitable") ou sfq, qui n'est pas vraiment le plus équitable, mais qui marche bien pour les grandes bandes passantes, sans utiliser trop de temps CPU. Il existe d'autres gestionnaires de file d'attente qui sont meilleurs, mais qui nécessitent plus de temps CPU. Le gestionnaire de mise en file d'attente Token Bucket Filter (filtre à seau de jetons) est souvent utilisé.

Maintenant, la seule chose qui reste à faire est de dire au noyau quels sont les paquets qui appartiennent à une classe. Nous le ferons tout d'abord nativement avec iproute2, mais des applications plus intéressantes sont possibles en combinaison avec netfilter.

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 match ip dst \
  150.151.23.24 flowid 10:200

# tc filter add dev eth0 parent 10:0 protocol ip prio 25 u32 match ip dst \
  150.151.0.0/16 flowid 10:100
```

Ici, on suppose que notre société (classe Office) est cachée derrière un pare-feu avec l'adresse IP 150.151.23.24 et que toutes nos autres adresses IP devront être considérées comme faisant partie de la classe ISP.

La classificateur u32 est un modèle très simple ; des règles de classification plus sophistiquées sont possibles lorsque l'on utilise netfilter pour marquer nos paquets. On peut ensuite utiliser ce marquage avec `tc`.

Maintenant, nous avons divisé équitablement la bande passante descendante, et nous avons besoin de faire la même chose avec le flux montant. Pour abréger, voici toutes les commandes en bloc :

```
# tc qdisc add dev eth1 root handle 20: cbq bandwidth 10Mbit avpkt 1000

# tc class add dev eth1 parent 20:0 classid 20:1 cbq bandwidth 10Mbit rate \
  10Mbit allot 1514 weight 1Mbit prio 8 maxburst 20 avpkt 1000

# tc class add dev eth1 parent 20:1 classid 20:100 cbq bandwidth 10Mbit rate \
  8Mbit allot 1514 weight 800Kbit prio 5 maxburst 20 avpkt 1000 \
  bounded

# tc class add dev eth1 parent 20:1 classid 20:200 cbq bandwidth 10Mbit rate \
  2Mbit allot 1514 weight 200Kbit prio 5 maxburst 20 avpkt 1000 \
  bounded

# tc qdisc add dev eth1 parent 20:100 sfq quantum 1514b perturb 15
# tc qdisc add dev eth1 parent 20:200 sfq quantum 1514b perturb 15

# tc filter add dev eth1 parent 20:0 protocol ip prio 100 u32 match ip src \
  150.151.23.24 flowid 20:200

# tc filter add dev eth1 parent 20:0 protocol ip prio 25 u32 match ip src \
  150.151.0.0/16 flowid 20:100
```

8.3 Que faire de la bande passante en excès ?

Dans notre exemple, il se trouve que même si les clients du fournisseur d'accès ne sont pas en majorité connectés (disons 8 heures du matin), nos bureaux n'ont toujours que 2 Mbits, ce qui est un peu du gaspillage.

En enlevant le paramètre "bounded", les classes pourront se prêter de la bande passante les unes aux autres.

Il se peut que des classes ne souhaitent pas prêter leur bande passante à d'autres. Deux fournisseurs d'accès rivaux sur un même lien peuvent ne pas vouloir s'offrir mutuellement de la bande passante pour des prunes. Dans ce cas, vous pouvez ajouter le mot-clé "isolated" à la fin de vos lignes "tc class add".

8.4 Subdivisions de classes

FIXME : suppositions qui n'ont pas du tout été testées ! Les essayer !

Nous pouvons aller plus loin. Supposons que tous les employés décident de lancer leur client "napster", il est toujours possible que notre base de données de routage dépasse la capacité de bande passante. Pour ce cas de figure, nous créons deux sous-classes, "Human" et "Database".

Notre base de données a toujours besoin de 500 Kbits, il nous reste donc 1,5 Mbits pour la consommation de la classe "Human".

Nous devons donc créer deux nouvelles classes à l'intérieur de la classe Office :

```
# tc class add dev eth0 parent 10:200 classid 10:250 cbq bandwidth 10Mbit rate \
  500Kbit allot 1514 weight 50Kbit prio 5 maxburst 20 avpkt 1000 \
  bounded

# tc class add dev eth0 parent 10:200 classid 10:251 cbq bandwidth 10Mbit rate \
  1500Kbit allot 1514 weight 150Kbit prio 5 maxburst 20 avpkt 1000 \
  bounded
```

FIXME : Finir cet exemple !

8.5 Équilibrage de charge sur plusieurs interfaces

FIXME : document TEQL

9 D'autres gestionnaires de mise en file d'attente (queueing disciplines)

Le noyau Linux nous offre beaucoup de gestionnaires de mises en file d'attente. Le plus largement utilisé est de loin la file d'attente pfifo_fast, qui est celle par défaut. Cela explique aussi pourquoi ces fonctionnalités avancées sont si robustes. Un autre modèle de file d'attente ne coûte rien en développement.

Chacune de ces files d'attente a ses forces et ses faiblesses. Toutes n'ont peut-être pas été bien testées.

9.1 pfifo_fast

Cette file d'attente, comme son nom l'indique (fifo = premier entré, premier sorti), signifie qu'il n'y a pas de traitement spécial pour les paquets reçus. En fait, ce n'est pas tout à fait vrai. Cette file d'attente a trois "bandes". À l'intérieur de chacune de ces bandes, la règle FIFO est appliquée. Cependant, tant qu'il y a un paquet en attente dans la "bande" 0, la "bande" 1 ne sera pas traitée. Il en va de même pour la "bande" 1 et la "bande" 2.

9.2 Stochastic Fairness Queueing

SFQ, comme dit précédemment, n'est pas vraiment déterministe, mais marche (en moyenne). Ses principaux avantages sont qu'elle a besoin de peu de CPU et de mémoire. Une véritable mise en file d'attente équitable nécessite que le noyau garde une trace de toutes les sessions courantes.

Stochastic Fairness Queueing (SFQ) est une implémentation simple de la famille des algorithmes de mise en file d'attente équitable. Cette implémentation est moins précise, mais nécessite aussi moins de calculs que les autres tout en étant presque parfaitement équitable.

Le concept clé dans SFQ est la conversation (ou flux), qui est une séquence de paquets de données ayant suffisamment de paramètres communs pour les distinguer des autres conversations. Les paramètres utilisés dans le cas de paquets IP sont les adresses de source et de destination, ainsi que le numéro de protocole.

SFQ consiste en l'allocation dynamique de files d'attente FIFO, une par conversation. Le gestionnaire de mise en file d'attente travaille en tourniquet (round-robin), envoyant un paquet de chaque FIFO à chaque tour. C'est pour cette raison qu'il est réputé équitable. Le principal avantage de SFQ est qu'il autorise le partage équitable du lien entre plusieurs applications et évite la monopolisation de la bande passante par un client. SFQ ne peut cependant pas distinguer les flux interactifs des flux de masse. On a, en général, recours à CBQ pour effectuer cette distinction, et diriger le flux de masse vers SFQ. [NdT : par flux de masse, il faut entendre gros flot de données, transmis en continu, comme un transfert FTP, par opposition à un flux interactif, comme celui généré par des requêtes HTTP].

9.3 Token Bucket Filter

Le Token Bucket Filter (TBF) est une file d'attente simple. Elle ne fait que passer les paquets entrants avec un taux de transfert dont les limites sont fixées administrativement. Il est possible de placer en mémoire tampon de courtes rafales de données.

L'implémentation TBF consiste en un tampon (seau), constamment rempli par des morceaux d'informations virtuelles appelés jetons, avec un débit spécifique (débit de jeton). Le paramètre le plus important du tampon est sa taille, qui est le nombre de jetons qu'il peut stocker.

Chaque jeton arrivant laisse sortir un paquet de données entrant de la file d'attente. Ce paquet est alors effacé du seau. L'association de cet algorithme avec les deux flux de jetons et de données, nous donne trois scénarios possibles :

- Les données arrivent dans TBF avec un débit *égal* au débit des jetons entrants. Dans ce cas, chaque paquet entrant a son jeton correspodant et passe la file d'attente sans délai.
- Les données arrivent dans TBF avec un débit *plus petit* que le débit des jetons. Seuls quelques jetons sont supprimés quand les paquets de données sortent de la file d'attente. Les jetons s'accumulent donc jusqu'à atteindre la taille du tampon. Les jetons en réserve peuvent être utilisés pour envoyer des données avec un débit supérieur au débit des jetons, si de courtes rafales de données arrivent.
- Les données arrivent dans TBF avec un débit *plus grand* que le débit des jetons. Dans ce cas, un dépassement de filtre a lieu. Les données entrantes peuvent être envoyées sans perte jusqu'à ce que les jetons accumulés soient utilisés. Après, les paquets au-dessus de la limite sont éliminés.

Le dernier scénario est très important, parce qu'il autorise la mise en forme administrative de la bande passante disponible pour les données traversant le filtre. L'accumulation de jetons autorise l'émission de courtes rafales de données sans perte, mais toute surcharge restante causera la perte systématique des paquets.

Le noyau Linux semble aller au-delà de cette spécification, et nous autorise aussi à limiter la vitesse de la transmission par rafales. Cependant, Alexey nous avertit :

```
Noter que le débit de pointe de TBF est beaucoup plus coriace : avec un MTU à 1500,
P_crit = 150 Koctets/s. Donc, si vous avez besoin d'un débit de pointe plus grand,
utilisez une alpha avec HZ=1000 :-)
```

FIXME : Est-ce encore vrai avec TSC (pentium+) ou équivalent ?

FIXME : Si ce n'est pas le cas, ajouter une section sur l'augmentation de fréquence

9.4 Random Early Detect

RED comporte quelques finesses supplémentaires. Quand une session TCP/IP démarre, on ne connaît pas la bande passante disponible. TCP/IP commence donc à transmettre lentement et va de plus en plus vite, mais est limité par le temps de latence au bout duquel les ACKs (acquittements) reviennent.

Une fois qu'un lien est saturé, RED commence à éliminer des paquets, ce qui indique à TCP/IP que le lien est congestionné, et qu'il devrait ralentir. La finesse réside dans le fait que RED simule la congestion réelle, et commence parfois à éliminer des paquets avant que le lien ne soit complètement saturé. Une fois que le lien est complètement saturé, il se comporte comme un contrôleur normal.

Pour plus d'informations sur le sujet, voir le chapitre Dorsale.

9.5 Ingress policer qdisc

Ingress qdisc sert si vous avez besoin de limiter le débit d'un hôte sans l'aide de routeurs ou d'autres machines Linux. Vous pouvez gérer la bande passante entrante et jeter des paquets quand cette bande passante dépasse le débit désiré. Cela peut, par exemple, préserver votre hôte d'une attaque SYN flood. Cela peut aussi servir à ralentir TCP/IP, qui réagit à la perte des paquets par la réduction de la vitesse.

FIXME : au lieu d'éliminer des paquets, pouvons-nous également les envoyer vers une vraie file d'attente ?

FIXME : la mise en forme en éliminant des paquets semble moins désirable que d'utiliser, par exemple, un filtre token bucket. Pas sûr cependant, les routeurs Cisco travaillent de cette manière, et les gens semblent en être contents.

Voir la référence 17 (IOS Committed Access Rate) à la fin de ce document.

En résumé, vous pouvez utiliser cela pour limiter la vitesse de téléchargement des fichiers et, donc, laisser plus de bande passante disponible pour les autres.

Voir la section sur la protection de votre hôte des attaques SYN flood pour un exemple de la manière dont cela marche.

10 Netfilter et iproute - marquage de paquets

Jusqu'à maintenant, nous avons vu comment iproute travaille, et netfilter a été mentionné plusieurs fois. Vous ne perdrez pas votre temps à consulter *Rusty's Remarkably Unreliable guides* `<http://netfilter.kernelnotes.org/unreliable-guides/>`.

Netfilter nous permet de filtrer les paquets, ou de désosser leurs en-têtes. Une de ses fonctionnalités particulières est de pouvoir marquer un paquet avec un nombre, grâce à l'option –set-mark.

Comme exemple, la commande suivante marque tous les paquets destinés au port 25, en l'occurence le courrier sortant.

```
# iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 25 \
 -j MARK --set-mark 1
```

Disons que nous avons plusieurs connexions, une qui est rapide (et chère, au mégaoctet) et une qui est plus lente, mais avec un tarif moins élevé. Nous souhaiterions que le courrier passe par la route la moins chère.

Nous avons déjà marqué le paquet avec un "1" et nous allons maintenant renseigner la base de données de la politique de routage pour qu'elle agisse sur ces paquets marqués.

```
# echo 201 mail.out >> /etc/iproute2/rt_tables
# ip rule add fwmark 1 table mail.out
# ip rule ls
0:      from all lookup local
32764:  from all fwmark        1 lookup mail.out
32766:  from all lookup main
32767:  from all lookup default
```

Nous allons maintenant générer la table mail.out avec une route vers la ligne lente, mais peu coûteuse.

```
# /sbin/ip route add default via 195.96.98.253 dev ppp0 table mail.out
```

Voilà qui est fait. Il se peut que nous voulions mettre en place des exceptions, et il y a beaucoup de moyens de le faire. Nous pouvons modifier la configuration de netfilter pour exclure certains hôtes, ou nous pouvons insérer une règle avec une priorité plus faible qui pointe sur la table principale pour nos hôtes faisant exception.

Nous pouvons aussi utiliser cette fonctionnalité pour nous conformer aux bits TOS en marquant les paquets avec différents types de service et les nombres correspondants. On crée ensuite les règles qui agissent sur ces types de service. De cette façon, on peut dédier une ligne RNIS aux connexions interactives.

Inutile de la dire, cela marche parfaitement sur un hôte qui fait de la translation d'adresse (NAT), autrement dit du "masquerading".

Note : pour ce faire, vous aurez besoin de valider quelques options du noyau :

```
IP: advanced router (CONFIG_IP_ADVANCED_ROUTER) [Y/n/?]
IP: policy routing (CONFIG_IP_MULTIPLE_TABLES) [Y/n/?]
IP: use netfilter MARK value as routing key (CONFIG_IP_ROUTE_FWMARK) [Y/n/?]
```

11 D'autres classificateurs

Les classificateurs sont les moyens par lesquels le noyau décide dans quelle file d'attente un paquet sera placé. Il y a divers classificateurs, chacun d'eux pouvant être utilisé pour différents buts.

fw
: Base la décision sur la façon dont la pare-feu a marqué les paquets.

u32
: Base la décision sur les champs à l'intérieur du paquet (c'est-à-dire l'adresse IP source, etc.)

route
: Base la décision sur la route à emprunter par le paquet.

rsvp, rsvp6
: Base la décision sur la cible (destination, protocole) et, optionnellement, sur la source (je pense).

tcindex
: FIXME : Remplissez-moi

Notez qu'il y a généralement plusieurs manières de classifier un paquet. Cela dépend du système de classification que vous souhaitez utiliser.

Les classificateurs acceptent en général quelques arguments communs. Ils sont listés ici pour des raisons pratiques :

protocol
: Le protocole que ce classificateur acceptera. Généralement, on n'acceptera que le trafic IP. Exigé.

parent
: La référence à laquelle ce classificateur est attaché. Cette référence doit être une classe déjà existante. Exigé.

prio
: La priorité de ce classificateur. Les plus grand nombres seront testés en premier.

handle

Cette référence a plusieurs significations suivant les différents filtres.

FIXME : En ajouter d'autres

Toutes les sections suivantes supposeront que vous essayez de mettre en forme le trafic allant vers `HostA`. Ces sections suppposeront que la classe racine a été configurée sur 1 : et que la classe vers laquelle vous voulez envoyer le trafic sélectionné est 1 :1.

11.1 Le classificateur "fw"

Le classificateur "fw" s'appuie sur le marquage des paquets à mettre en forme par le pare-feu. Donc, nous configurerons d'abord le pare-feu pour les marquer :

```
# iptables -I PREROUTING -t mangle -p tcp -d HostA \
 -j MARK --set-mark 1
```

Maintenant, tous les paquets vers cette machine (HostA) sont balisés avec la marque 1. À présent, nous construisons les règles de mise en forme pour mettre en forme les paquets. Nous avons juste besoin d'indiquer que les paquets balisés avec la marque 1 vont vers la classe 1 :1. C'est fait par la commande suivante :

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 1 handle 1 fw classid 1:1
```

Cela devrait se comprendre de soi-même. On attache à la classe 1 :0 un filtre avec la priorité 1 pour filtrer tous les paquets marqués à 1 par le pare-feu vers la classe 1 :1. Noter l'utilisation du paramètre "handle" pour indiquer la marque attendue.

C'est tout ce qu'il y a à faire ! C'est le procédé le plus simple (à mon humble avis). Je pense que les autres procédés sont plus difficiles à comprendre. Notez que toute la puissance du pare-feu peut être utilisée avec ce classificateur. Cela inclut l'analyse des adresses MAC, des identificateurs d'utilisateurs (user ID) et tout ce que le firewall peut traiter.

11.2 Le classificateur "u32"

Le filtre u32 est le filtre le plus avancé dans l'implémentation courante. Il est entièrement basé sur des tables de hachage, ce qui le rend robuste quand il y a beaucoup de règles de filtrage.

Dans sa forme la plus simple, le filtre u32 est une liste d'enregistrements, chacun consistant en deux champs : un sélecteur et une action. Les sélecteurs, décrits ci-dessous, sont comparés avec le paquet IP traité jusqu'à la première correspondance, et l'action associée est accomplie. Le type d'action le plus simple serait de diriger le paquet vers une classe CBQ définie.

La ligne de commande du programme filtre `tc`, utilisé pour configurer le filtre, consiste en trois parties : la spécification du filtre, un sélecteur et une action. La spécification du filtre peut être définie comme :

```
tc filter add dev IF [ protocol PROTO ]
                     [ (preference|priority) PRIO ]
                     [ parent CBQ ]
```

Le champ `protocol` décrit le protocole sur lequel le filtre sera appliqué. Nous ne discuterons que du cas du protocole `ip`. Le champ `preference`(`priority` peut être utilisé comme alternative) fixe la priorité du filtre que l'on définit. C'est important dans la mesure où vous pouvez avoir plusieurs filtres (listes de règles) avec des priorités différentes. Chaque liste sera scrutée dans l'ordre d'ajout des règles. Alors, la liste avec la priorité la plus faible (celle qui a le numéro de préférence le plus élevé) sera traitée. Le champ `parent` définit le sommet de l'arbre CBQ (par ex. 1 :0) auquel le filtre doit être attaché.

Les options décrites s'appliquent à tous les filtres, pas seulement à u32.

11.2.1 Le sélecteur U32

Le sélecteur U32 contient la définition d'un modèle, qui sera comparé au paquet traité. Plus précisément, il définit quels bits doivent correspondre dans l'en-tête du paquet, et rien de plus, mais cette méthode simple est très puissante. Jetons un oeil sur l'exemple suivant, directement tiré d'un filtre assez complexe réellement existant :

```
# filter parent 1: protocol ip pref 10 u32 fh 800::800 order 2048 key ht 800 bkt 0 flowid 1:3 \
  match 00100000/00ff0000 at 0
```

Pour l'instant, laissons de côté la première ligne - tous ces paramètres décrivent les tables de hachage du filtre. Focalisons-nous sur la ligne de sélection contenant le mot-clé `match`. Ce sélecteur fera correspondre les en-têtes IP dont le second octet sera 0x10 (0010). Comme nous pouvons le deviner, le nombre 00ff est le masque de correspondance, disant au filtre quels bits il doit regarder. Ici, c'est 0xff, donc l'octet correspondra si c'est exactement 0x10. Le mot-clé `at` signifie que la correspondance doit démarrer au décalage spécifié (en octets) - dans notre cas, c'est au début du paquet. Traduisons tout cela en langage humain : le paquet correspondra si son champ Type de Service (TOS) a le bit "faible délai" positionné. Analysons une autre règle :

```
# filter parent 1: protocol ip pref 10 u32 fh 800::803 order 2051 key ht 800 bkt 0 flowid 1:3 \
  match 00000016/0000ffff at nexthdr+0
```

L'option `nexthdr` désigne l'en-tête suivant encapsulé dans le paquet IP, c'est à dire celui du protocole de la couche supérieure. La correspondance commencera également au début du prochain en-tête. Elle devrait avoir lieu dans le deuxième mot de 32 bits de l'en-tête. Dans les protocoles TCP et UDP, ce champ contient le port de destination du paquet. Le nombre est donné dans le format big-endian, c'est-à-dire les bits les plus significatifs en premier. Il faut donc lire 0x0016 comme 22 en décimal, qui correspond au service SSH dans le cas de TCP. Comme vous le devinez, cette correspondance est ambiguë sans un contexte, et nous en discuterons plus loin.

Ayant compris tout cela, nous trouverons le sélecteur suivant très facile à lire : `match c0a80100/ffffff00 at 16`. Ce que nous avons ici, c'est une correspondance de trois octets au 17ème octet, en comptant à partir du début de l'en-tête IP. Cela correspond aux paquets qui ont une adresse de destination quelconque dans le réseau 192.168.1/24. Après avoir analysé les exemples, nous pouvons résumer ce que nous avons appris.

11.2.2 Sélecteurs généraux

Les sélecteurs généraux définissent le modèle, le masque et le décalage qui seront comparés au contenu du paquet. En utilisant les sélecteurs généraux, vous pouvez rechercher des correspondances sur n'importe quel bit de l'en-tête IP (ou des couches supérieures). Ils sont quand même plus difficiles à écrire et à lire que les sélecteurs spécifiques décrits ci-dessus. La syntaxe générale des sélecteurs est :

```
match [ u32 | u16 | u8 ] PATTERN MASK [ at OFFSET | nexthdr+OFFSET]
```

Un des mots-clés `u32`,`u16` ou `u8` doit spécifier la longueur du modèle en bits. PATTERN et MASK se rapporteront à la longueur définie par ce mot-clé. Le paramètre OFFSET est le décalage, en octets, pour le démarrage de la recherche de correspondance. Si le mot-clef `nexthdr+` est présent, le décalage sera relatif à l'en-tête de la couche réseau supérieure.

Quelques exemples :

```
# tc filter add dev ppp14 parent 1:0 prio 10 u32 \
     match u8 64 0xff at 8 \
     flowid 1:4
```

Un paquet correspondra à cette règle si sa "durée de vie" (TTL) est de 64. TTL est le champ démarrant juste après le 8ème octet de l'en-tête IP.

```
# tc filter add dev ppp14 parent 1:0 prio 10 u32 \
     match u8 0x10 0xff at nexthdr+13 \
     protocol tcp \
     flowid 1:3 \
```

Cette règle correspondra seulement aux paquets TCP avec le bit ACK positionné. Ici, nous pouvons voir un exemple d'utilisation de deux sélecteurs, le résultat final étant un ET logique de leurs résultats. Si vous jetez un oeil sur un schéma de l'en-tête TCP, vous pouvez voir que le bit ACK est le second bit (0x10) du 14ème octet de l'en-tête TCP (`at nexthdr+13`). Comme second sélecteur, si vous voulez vous compliquer la vie, vous pouvez écrire `match u8 0x06 0xff at 9` à la place du sélecteur spécifique `protocol tcp`, puisque 6 est le numéro du protocole TCP, spécifié au 10ème octet de l'en-tête IP. En revanche, dans cet exemple, vous ne pourrez pas utiliser de sélecteur spécifique pour la première correspondance, simplement parce qu'il n'y a pas de sélecteur spécifique pour désigner les bits TCP ACK.

11.2.3 Les sélecteurs spécifiques

La table suivante contient la liste de tous les sélecteurs spécifiques que les auteurs de cette section ont trouvés dans le code source du programme `tc`. Ils rendent simplement la vie plus facile en accroissant la lisibilité de la configuration du filtre.

FIXME : emplacement de la table - la table est dans un fichier séparé "selector.html"

FIXME : C'est encore en Polonais :-(FIXME : doit être "sgmlisé"

Quelques exemples :

```
# tc filter add dev ppp0 parent 1:0 prio 10 u32 \
     match ip tos 0x10 0xff \
     flowid 1:4
```

La règle ci-dessus correspondra à des paquets qui ont le champ TOS égal à 0x10. Le champ TOS commence au deuxième octet du paquet et occupe 1 octet, ce qui nous permet d'écrire un sélecteur général équivalent : `match u8 0x10 0xff at 1`. Cela nous donne une indication sur l'implémentation du filtre u32 ; les règles spécifiques sont toujours traduites en règles générales, et c'est sous cette forme qu'elles sont stockées en mémoire par le noyau. Cela amène à une autre conclusion : les sélecteurs `tcp` et `udp` sont exactement les mêmes et c'est la raison pour laquelle vous ne pouvez pas utiliser un simple sélecteur `match tcp dst 53 0xffff` pour désigner un paquet TCP envoyé sur un port donné : cela désigne aussi les paquets UDP envoyés sur ce port. Vous devez également spécifier le protocole avec la règle suivante :

```
# tc filter add dev ppp0 parent 1:0 prio 10 u32 \
        match tcp dst 53 0xffff \
        match ip protocol 0x6 0xff \
        flowid 1:2
```

11.3 Le classificateur "route"

Ce classificateur filtre en se basant sur le résultat des tables de routage. Quand un paquet passant à travers les classes en atteint une qui est marquée avec le filtre "route", il divise le paquet en se basant sur l'information de la table de routage.

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 route
```

Ici, nous ajoutons un classificateur `route` sur le noeud parent 1 :0 avec la priorité 100. Quand un paquet atteint ce noeud (ce qui, puisqu'il est racine, arrive immédiatement), il consulte la table de routage et si une entrée de la table correspond, il envoie le paquet vers la classe donnée et lui donne une priorité de 100. Ensuite, pour finalement activer les choses, vous ajoutez l'entrée de routage appropriée.

L'astuce ici est de définir "realm" en se basant soit sur la destination, soit sur la source. Voici la façon de faire cela :

```
# ip route add Host/Network via Gateway dev Device realm RealmNumber
```

Par exemple, nous pouvons définir notre réseau de destination 192.168.10.0 avec le nombre "realm" égal à 10 :

```
# ip route add 192.168.10.0/24 via 192.168.10.1 dev eth1 realm 10
```

Quand on ajoute des filtres "route", on peut utiliser les nombres "realm" pour représenter les réseaux ou les hôtes et spécifier quelle est la correspondance entre les routes et les filtres.

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 \
  route to 10 classid 1:10
```

La règle ci-dessus indique que les paquets allant vers le réseau 192.168.10.0 correspondent à la classe 1 :10.

Le filtre route peut aussi être utilisé avec les routes sources. Par exemple, il y a un sous-réseau attaché à notre routeur Linux sur eth2.

```
# ip route add 192.168.2.0/24 dev eth2 realm 2
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 \
  route from 2 classid 1:2
```

Ici, le filtre spécifie que les paquets venant du réseau 192.168.2.0 (realm 2) correspondront à la classe 1 :2.

11.4 Le classificateur "rsvp"

FIXME : à remplir

11.5 Le classificateur "tcindex"

FIXME : à remplir

12 Paramètres réseau du noyau

Le noyau utilise de nombreux paramètres qui peuvent être ajustés en différentes circonstances. Bien que, comme d'habitude, les paramètres par défaut conviennent à 99% des installations, nous ne pourrions pas appeler ce document "HOWTO avancé" sans en dire un mot.

Les éléments intéressants sont dans /proc/sys/net, jetez-y un oeil. Tout ne sera pas documenté ici au départ, mais nous y travaillons.

12.1 Filtrage du Chemin Inverse (Reverse Path Filtering)

Par défaut, les routeurs routent tout, même les paquets qui visiblement n'appartiennent pas à votre réseau. Un exemple courant est l'espace des adresses IP privées s'échappant sur Internet. Si vous avez une interface avec une route pour 195.96.96.0/24 dessus, vous ne vous attendrez pas à voir arriver des paquets venant de 212.64.94.1.

Beaucoup d'utilisateurs veulent désactiver cette fonctionnalité. Les développeurs du noyau ont permis de le faire facilement. Il y a des fichiers dans `/proc` où vous pouvez ordonner au noyau de le faire pour vous. La méthode est appelée "Reverse Path Filtering" (Filtrage par chemin inverse). Pour faire simple, si la réponse à ce paquet ne sort pas par l'interface par laquelle il est entré, alors c'est un paquet "bogué" et il sera ignoré.

Les instructions suivantes vont activer cela pour toutes les interfaces courantes et futures.

```
# for i in /proc/sys/net/ipv4/conf/*/rp_filter ; do
>  echo 2 > $i
> done
```

En reprenant l'exemple du début, si un paquet arrivant sur le routeur Linux par eth1 prétend venir du réseau Bureau+FAI, il sera éliminé. De même, si un paquet arrivant du réseau Bureau prétend être de quelque part à l'extérieur du pare-feu, il sera également éliminé.

Ce qui est présenté ci-dessus est le filtrage de chemin inverse complet. Le paramétrage par défaut filtre seulement sur les adresses IP des réseaux directement connectés. Ce paramétrage par défaut est utilisé parce que le filtrage complet échoue dans le cas d'un routage asymétrique (où il y a des paquets arrivant par un chemin et ressortant par un autre, comme dans le cas du trafic satellite, ou si vous avez des routes dynamiques (bgp, ospf, rip) dans votre réseau. Les données descendent vers la parabole satellite et les réponses repartent par des lignes terrestres normales).

Si cette exception s'applique dans votre cas (vous devriez être au courant), vous pouvez simplement désactiver le `rp\_filter` sur l'interface d'arrivée des données satellite. Si vous voulez voir si des paquets sont éliminés, le fichier `log\_martians` du même répertoire indiquera au noyau de les enregistrer dans votre syslog.

```
# echo 1 >/proc/sys/net/ipv4/conf/<interfacename>/log_martians
```

FIXME : Est-ce que la configuration des fichiers dans .../conf/{default,all} suffit ? - martijn

12.2 Configurations obscures

Bon, il y a beaucoups de paramètres qui peuvent être modifiés. Nous essayons de tous les lister. Voir aussi une documentation partielle dans `Documentation/ip-sysctl.txt`.

Certaines de ces configurations ont des valeurs par défaut différentes suivant que vous répondez "Yes" ou "No" à la question "Configure as router and not host" lors de la compilation du noyau.

12.2.1 ipv4 générique

En remarque générale, les fonctionnalités de limitation de débit ne fonctionnent pas sur loopback, donc n'essayez pas de les tester localement.

/proc/sys/net/ipv4/icmp_destunreach_rate

FIXME : à remplir

/proc/sys/net/ipv4/icmp_echo_ignore_all

FIXME : à remplir

/proc/sys/net/ipv4/icmp_echo_ignore_broadcasts [Useful]
Si vous pinguez l'adresse de diffusion d'un réseau, tous les hôtes sont censés répondre. Cela permet de coquettes attaques de déni de service. Mettez cette valeur à 1 pour ignorer ces messages de diffusion.

/proc/sys/net/ipv4/icmp_echoreply_rate
FIXME : à remplir

/proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
FIXME : à remplir

/proc/sys/net/ipv4/icmp_paramprob_rate
FIXME : à remplir

/proc/sys/net/ipv4/icmp_timeexceed_rate
Voici la célèbre cause des "étoiles Solaris" dans traceroute. Limite le nombre de messages ICMP Time Exceeded envoyés. FIXME : Unité de ces "rates" ? Soit je suis stupide, soit ça ne fonctionne pas.

/proc/sys/net/ipv4/igmp_max_memberships
FIXME : à remplir

/proc/sys/net/ipv4/inet_peer_gc_maxtime
FIXME : à remplir

/proc/sys/net/ipv4/inet_peer_gc_mintime
FIXME : à remplir

/proc/sys/net/ipv4/inet_peer_maxttl
FIXME : à remplir

/proc/sys/net/ipv4/inet_peer_minttl
FIXME : à remplir

/proc/sys/net/ipv4/inet_peer_threshold
FIXME : à remplir

/proc/sys/net/ipv4/ip_autoconfig
FIXME : à remplir

/proc/sys/net/ipv4/ip_default_ttl
Durée de vie (TTL) des paquets. Fixez-la à la valeur sûre de 64. Augmentez-la si vous avez un réseau immense, mais pas "pour voir" : les boucles sans fin d'un mauvais routage sont plus dangereuses si le TTL est élevé. Vous pouvez même envisager de diminuer la valeur dans certaines circonstances.

/proc/sys/net/ipv4/ip_dynaddr
Vous aurez besoin de positionner cela si vous utilisez la connexion à la demande avec une adresse d'interface dynamique. Une fois que votre interface sera activée, tout paquet placé en file d'attente sera retraité pour avoir la bonne adresse. Cela résout le problème posé par une connexion défectueuse ayant configuré une interface, suivie par une deuxième tentative réussie (avec une adresse IP différente).

/proc/sys/net/ipv4/ip_forward
Le noyau doit-il essayer de transmettre les paquets ? Désactivé par défaut pour les hôtes, activé par défaut quand le noyau est configuré pour un routeur.

/proc/sys/net/ipv4/ip_local_port_range
Intervalle des ports locaux pour les connections sortantes. En fait, assez petit par défaut, 1024 à 4999.

/proc/sys/net/ipv4/ip_no_pmtu_disc
Configurez cela si vous voulez désactiver la découverte du MTU : une technique pour déterminer le plus grand MTU possible sur votre chemin.

/proc/sys/net/ipv4/ipfrag_high_thresh
FIXME : à remplir

/proc/sys/net/ipv4/ipfrag_low_thresh
FIXME : à remplir

/proc/sys/net/ipv4/ipfrag_time
FIXME : à remplir

/proc/sys/net/ipv4/tcp_abort_on_overflow
FIXME : à remplir

/proc/sys/net/ipv4/tcp_fin_timeout
FIXME : à remplir

/proc/sys/net/ipv4/tcp_keepalive_intvl
FIXME : à remplir

/proc/sys/net/ipv4/tcp_keepalive_probes
FIXME : à remplir

/proc/sys/net/ipv4/tcp_keepalive_time
FIXME : à remplir

/proc/sys/net/ipv4/tcp_max_orphans
FIXME : à remplir

/proc/sys/net/ipv4/tcp_max_syn_backlog
FIXME : à remplir

/proc/sys/net/ipv4/tcp_max_tw_buckets
FIXME : à remplir

/proc/sys/net/ipv4/tcp_orphan_retries
FIXME : à remplir

/proc/sys/net/ipv4/tcp_retrans_collapse
FIXME : à remplir

/proc/sys/net/ipv4/tcp_retries1
FIXME : à remplir

/proc/sys/net/ipv4/tcp_retries2
FIXME : à remplir

/proc/sys/net/ipv4/tcp_rfc1337
FIXME : à remplir

/proc/sys/net/ipv4/tcp_sack
Utilise un ACK sélectif qui peut être utilisé pour signifier qu'un seul paquet est manquant. Facilite ainsi une récupération rapide.

/proc/sys/net/ipv4/tcp_stdurg
FIXME : à remplir

/proc/sys/net/ipv4/tcp_syn_retries
FIXME : à remplir

/proc/sys/net/ipv4/tcp_synack_retries
FIXME : à remplir

/proc/sys/net/ipv4/tcp_timestamps
FIXME : à remplir

/proc/sys/net/ipv4/tcp_tw_recycle
FIXME : à remplir

/proc/sys/net/ipv4/tcp_window_scaling
TCP/IP autorise normalement des fenêtres jusqu'à une taille de 65535 octets. Pour des réseaux vraiment rapides, cela peut ne pas être assez. Les options "windows scaling" autorisent des fenêtres jusqu'au gigaoctet, ce qui est bon pour les produits à grande bande passante mais fort délai.

12.2.2 Configuration périphérique par périphérique

DEV peut désigner soit une interface réelle, soit "all", soit "default". Default change également les paramètres des interfaces qui seront créées par la suite.

/proc/sys/net/ipv4/conf/DEV/accept_redirects
Si un routeur décide que vous l'utilisez à tort (c'est-à-dire qu'il a besoin de ré-envoyer votre paquet sur la même interface), il vous enverra un ICMP Redirect. Cela présente cependant un petit risque pour la sécurité, et vous pouvez le désactiver, ou utiliser les redirections sécurisées.

/proc/sys/net/ipv4/conf/DEV/accept_source_route
Plus vraiment utilisé. On l'utilisait pour être capable de donner à un paquet une liste d'adresses IP à visiter. Linux peut être configuré pour satisfaire cette option IP.

/proc/sys/net/ipv4/conf/DEV/bootp_relay
FIXME : à remplir

/proc/sys/net/ipv4/conf/DEV/forwarding
FIXME : à remplir

/proc/sys/net/ipv4/conf/DEV/log_martians
Voir la section sur le "reverse path filters"

/proc/sys/net/ipv4/conf/DEV/mc_forwarding
Si vous faites de la transmission multidiffusion (multicast) sur cette interface.

/proc/sys/net/ipv4/conf/DEV/proxy_arp
FIXME : à remplir

/proc/sys/net/ipv4/conf/DEV/rp_filter
Voir la section sur le "reverse path filters"

/proc/sys/net/ipv4/conf/DEV/secure_redirects
FIXME : à remplir

/proc/sys/net/ipv4/conf/DEV/send_redirects
Si vous employez les redirections mentionnées ci-dessus.

/proc/sys/net/ipv4/conf/DEV/shared_media
FIXME : à remplir

/proc/sys/net/ipv4/conf/DEV/tag
FIXME : à remplir

12.2.3 Politique de voisinage

DEV peut désigner soit une interface réelle, soit "all", soit "default". Default change également les paramètres des interfaces qui seront créées par la suite.

/proc/sys/net/ipv4/neigh/DEV/anycast_delay
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/app_solicit
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/base_reachable_time
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/delay_first_probe_time
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/gc_stale_time
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/locktime
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/mcast_solicit
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/proxy_delay
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/proxy_qlen
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/retrans_time
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/ucast_solicit
FIXME : à remplir

/proc/sys/net/ipv4/neigh/DEV/unres_qlen
FIXME : à remplir

12.2.4 Configuration du routage

/proc/sys/net/ipv4/route/error_burst
FIXME : à remplir

/proc/sys/net/ipv4/route/error_cost
FIXME : à remplir

/proc/sys/net/ipv4/route/flush
FIXME : à remplir

/proc/sys/net/ipv4/route/gc_elasticity
FIXME : à remplir

/proc/sys/net/ipv4/route/gc_interval
FIXME : à remplir

/proc/sys/net/ipv4/route/gc_min_interval
FIXME : à remplir

/proc/sys/net/ipv4/route/gc_thresh
FIXME : à remplir

/proc/sys/net/ipv4/route/gc_timeout
FIXME : à remplir

/proc/sys/net/ipv4/route/max_delay
FIXME : à remplir

/proc/sys/net/ipv4/route/max_size
FIXME : à remplir

/proc/sys/net/ipv4/route/min_adv_mss
FIXME : à remplir

/proc/sys/net/ipv4/route/min_delay
FIXME : à remplir

/proc/sys/net/ipv4/route/min_pmtu
FIXME : à remplir

/proc/sys/net/ipv4/route/mtu_expires
FIXME : à remplir

/proc/sys/net/ipv4/route/redirect_load
FIXME : à remplir

/proc/sys/net/ipv4/route/redirect_number
FIXME : à remplir

/proc/sys/net/ipv4/route/redirect_silence
FIXME : à remplir

13 Application du contrôle de trafic aux dorsales

Ce chapitre est conçu comme une introduction au routage de dorsales (backbones). Ces liaisons impliquent souvent des bandes passantes supérieures à 100 mégabits/s, ce qui nécessite une approche différente de celle de votre modem ADSL à la maison.

13.1 Files d'attente de routeurs

Le comportement normal des files d'attente de routeurs est appelé "tail-drop" (NdT : élimine le reste). Le "tail-drop" consiste à mettre en file d'attente un certain volume de trafic et à éliminer tout ce qui déborde. Ce n'est pas du tout équitable et cela conduit à des retransmissions de synchronisation. Quand une retransmission de synchronisation a lieu, la brusque rafale de rejet d'un routeur qui a atteint sa limite entraînera une rafale de retransmissions retardée qui inondera à nouveau le routeur congestionné.

Dans le but d'en finir avec les congestions occasionnelles des liens, les routeurs de dorsales intègrent souvent des files d'attente de grande taille. Malheureusement, bien que ces files d'attente offrent un bon débit, elles peuvent augmenter sensiblement les temps de latence et entraîner un comportement très saccadé des connections TCP pendant la congestion.

Ces problèmes avec le "tail-drop" deviennent de plus en plus préoccupants avec l'augmentation de l'utilisation d'applications hostiles au réseau. Le noyau Linux nous offre la technique RED, abréviation de Random Early Detect, ou détection précoce directe.

RED n'est pas la solution miracle à tous ces problèmes. Les applications qui n'intègrent pas correctement la technique de "l'exponential backoff" obtiennent toujours une part trop grande de bande passante. Cependant, avec la technique RED elles ne provoquent pas trop de dégâts sur le débit et les temps de latence des autres connexions.

RED élimine statistiquement des paquets du flux avant qu'il n'atteigne sa limite "dure" (hard). Sur une dorsale congestionnée, cela entraîne un ralentissement en douceur de la liaison et évite les retransmissions de synchronisation. La technique RED aide aussi TCP à trouver une vitesse "équitable" plus rapidement : en permettant d'éliminer des paquets plus tôt, il conserve une file d'attente plus courte et des temps de latence mieux contrôlés. La probabilité qu'un paquet soit éliminé d'une connexion particulière est proportionnelle à la bande passante utilisée par cette connexion plutôt qu'au nombre de paquets qu'elle envoie.

La technique RED est une bonne gestion de file d'attente pour les dorsales, où vous ne pouvez pas vous permettre le coût d'une mémorisation d'état par session qui est nécessaire pour une mise en file d'attente vraiment équitable.

Pour utiliser RED, vous devez régler trois paramètres : Min, Max et burst. Min est la taille minimum de la file d'attente en octets avant que les rejets n'aient lieu, Max est le maximum "doux" (soft) en-dessous duquel l'algorithme s'efforcera de rester, et burst est le nombre maximum de paquets envoyés "en rafale".

Vous devriez configurer Min en calculant le plus grand temps de latence acceptable pour la mise en file d'attente, multiplié par votre bande passante. Par exemple, sur mon lien ISDN à 64 Kbits/s, je voudrais avoir un temps de latence de base de mise en file d'attente de 200 ms. Je configure donc Min à 1600 octets (= 0,2 x 64000 / 8). Imposer une valeur Min trop petite va dégrader le débit et une valeur Min trop grande va dégrader le temps de latence. Sur une liaison lente, choisir un coefficient Min petit ne peut pas remplacer une réduction du MTU pour améliorer les temps de réponse.

Vous devriez configurer Max à au moins deux fois Min pour éviter les synchronisations. Sur des liens lents avec de petites valeurs de Min, il peut être prudent d'avoir Max quatre fois plus grand que Min ou plus.

Burst contrôle la réponse de l'algorithme RED aux rafales. Burst doit être choisi plus grand que min/avpkt (paquet moyen). Expérimentalement, j'ai trouvé que (min+min+max)/(3*avpkt) marche bien.

De plus, vous devez configurer limit et avpkt. Limit est une valeur de sécurité : s'il y a plus de Limit octets dans la file, RED reprend la technique "tail-drop". Je choisis une valeur typique égale à 8 fois Max. Avpkt devrait être fixé à la taille moyenne d'un paquet. 1000 fonctionne correctement sur des liaisons Internet haut débit ayant un MTU de 1500 octets.

Lire *the paper on RED queueing* `<http://www.aciri.org/floyd/papers/red/red.html>` par Sally Floyd et Van Jacobson pour les informations techniques.

FIXME : besoin de plus d'infos. Cela dépend de toi, Greg :-) - ahu

14 Recettes de cuisine pour la mise en forme du trafic

Cette section contient des "recettes de cuisine" qui peuvent vous aider à résoudre vos problèmes. Un livre de cuisine ne remplace cependant pas une réelle compréhension, essayez donc d'assimiler ce qui suit.

14.1 Faire tourner plusieurs sites avec différentes SLA (autorisations)

Vous pouvez faire cela de plusieurs manières. Apache possède un module qui permet de le supporter, mais nous montrerons comment Linux peut le faire pour d'autres services. Les commandes ont été reprises d'une présentation de Jamal Hadi, dont la référence est fournie ci-dessous.

Disons que nous avons deux clients, avec http, ftp et du streaming audio, et que nous voulions leur vendre une largeur de bande passante limitée. Nous le ferons sur le serveur lui-même.

Le client A doit disposer d'au moins 2 mégabits, et le client B a payé pour 5 mégabits. Nous séparons nos clients en créant deux adresses IP virtuelles sur notre serveur.

```
# ip address add 188.177.166.1 dev eth0
# ip address add 188.177.166.2 dev eth0
```

C'est à vous d'associer les différents serveurs à la bonne adresse IP. Tous les démons courants supportent cela.

Nous pouvons tout d'abord attacher une mise en file d'attente CBQ à eth0 :

```
# tc qdisc add dev eth0 root handle 1: bandwidth 10Mbit cell 8 avpkt 1000 \
  mpu 64
```

Nous créons ensuite les classes pour nos clients :

```
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 10Mbit rate \
  2MBit avpkt 1000 prio 5 bounded isolated allot 1514 weight 1 maxburst 21
# tc class add dev eth0 parent 1:0 classid 1:2 cbq bandwidth 10Mbit rate \
  5Mbit avpkt 1000 prio 5 bounded isolated allot 1514 weight 1 maxburst 21
```

Nous ajoutons les filtres pour nos deux classes :

```
##FIXME: Pourquoi cette ligne, que fait-elle ? Qu'est-ce qu'un
diviseur ?
##FIXME: Un diviseur est lié à une table de hachage et au nombre de
seaux -ahu
# tc filter add dev eth0 parent 1:0 protocol ip prio 5 handle 1: u32 divisor 1
# tc filter add dev eth0 parent 1:0 prio 5 u32 match ip src 188.177.166.1
  flowid 1:1
# tc filter add dev eth0 parent 1:0 prio 5 u32 match ip src 188.177.166.2
  flowid 1:2
```

Et voilà qui est fait.

FIXME : Pourquoi pas un filtre token bucket ? Y a t-il un retour par défaut à pfifo_fast quelque part ?

14.2 Protéger votre machine des inondations SYN floods

D'après la documentation iproute d'Alexeys, adaptée à netfilter. Si vous utilisez ceci, prenez garde d'ajuster les nombres avec des valeurs raisonnables pour votre système.

Si vous voulez protéger tout un réseau, oubliez ce script, qui est plus adapté à un hôte seul.

```
#! /bin/sh -x
#
# script simple utilisant les capacités de Ingress
# Ce script montre comment on peut limiter le flux entrant des SYN.
# Utile pour la protection des TCP-SYN. Vous pouvez utiliser IPchains
# pour bénéficier de puissantes fonctionnalités sur les SYN.
#
# chemins vers les divers utilitaires
# À changer en fonction des vôtres
#
TC=/sbin/tc
IP=/sbin/ip
IPTABLES=/sbin/iptables
INDEV=eth2
#
# marque tous les paquets SYN entrant à travers $INDEV avec la valeur 1
############################################################
$iptables -A PREROUTING -i $INDEV -t mangle -p tcp --syn \
  -j MARK --set-mark 1
############################################################
#
# installe la file d'attente ingress sur l'interface associée
############################################################
```

```
$TC qdisc add dev $INDEV handle ffff: ingress
##############################################################
#
# Les paquets SYN ont une taille de 40 octets (320 bits), donc trois SYN
# ont une taille de 960 bits (approximativement 1Kbit) ; nous limitons donc
# les SYNs entrants à 3 par seconde (pas vraiment utile, mais sert à
# montrer ce point -JHS
##############################################################
$TC filter add dev $INDEV parent ffff: protocol ip prio 50 handle 1 fw \
police rate 1kbit burst 40 mtu 9k drop flowid :1
##############################################################


#
echo "---- qdisc parameters Ingress  ----------"
$TC qdisc ls dev $INDEV
echo "---- Class parameters Ingress  ----------"
$TC class ls dev $INDEV
echo "---- filter parameters Ingress ----------"
$TC filter ls dev $INDEV parent ffff:

#supprime la file d'attente ingress
#$TC qdisc del $INDEV ingress
```

14.3 Limiter le débit ICMP pour empêcher les dénis de service

Récemment, les attaques distribuées de déni de service sont devenues une nuisance importante sur Internet. En filtrant proprement et en limitant le débit de votre réseau, vous pouvez à la fois éviter de devenir victime ou source de ces attaques.

Vous devriez filtrer vos réseaux de telle sorte que vous n'autorisiez pas les paquets avec une adresse IP source non-locale à quitter votre réseau. Cela empêche les utilisateurs d'envoyer de manière anonyme des cochonneries sur Internet.

La limitation de débit peut faire encore mieux, comme vu plus haut. Pour vous rafraîchir la mémoire, revoici notre diagramme ASCII :

```
[Internet] ---<E3, T3, n'importe quoi>--- [routeur Linux] --- [Bureau+FAI]
                                          eth1          eth0
```

Nous allons d'abord configurer les parties prérequises :

```
# tc qdisc add dev eth0 root handle 10: cbq bandwidth 10Mbit avpkt 1000
# tc class add dev eth0 parent 10:0 classid 10:1 cbq bandwidth 10Mbit rate \
  10Mbit allot 1514 prio 5 maxburst 20 avpkt 1000
```

Si vous avez des interfaces de 100 Mbits ou plus, ajustez ces nombres. Maintenant, vous devez déterminer combien de trafic ICMP vous voulez autoriser. Vous pouvez réaliser des mesures avec tcpdump, en écrivant les résultats dans un fichier pendant un moment, et regarder combien de paquets ICMP passent par votre réseau. Ne pas oublier d'augmenter la longueur du "snapshot". Si la mesure n'est pas possible, vous pouvez consacrer par exemple 5% de votre bande passante disponible. Configurons notre classe :

```
# tc class add dev eth0 parent 10:1 classid 10:100 cbq bandwidth 10Mbit rate \
  100Kbit allot 1514 weight 800Kbit prio 5 maxburst 20 avpkt 250 \
  bounded
```

Cela limite le débit à 100 Kbits sur la classe. Maintenant, nous avons besoin d'un filtre pour assigner le trafic ICMP à cette classe :

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 match ip
  protocol 1 0xFF flowid 10:100
```

14.4 Donner la priorité au trafic interactif

Si beaucoup de données arrivent à votre lien ou en partent, et que vous essayez de faire de la maintenance via telnet ou ssh, cela peut poser problème : d'autres paquets bloquent vos frappes clavier. Cela ne serait-il pas mieux si vos paquets interactifs pouvaient se faufiler dans le trafic de masse ? Linux peut faire cela pour vous.

Comme précédemment, nous avons besoin de manipuler le trafic dans les deux sens. Evidemment, cela marche mieux s'il y a des machines Linux aux deux extrémités du lien, bien que d'autres UNIX soient capables de faire la même chose. Consultez votre gourou local Solaris/BSD pour cela.

Le gestionnaire standard pfifo_fast a trois "bandes" différentes. Le trafic de la bande 0 est transmis en premier, le trafic des bandes 1 et 2 étant traité après. Il est vital que votre trafic interactif soit dans la bande 0 ! Ce qui suit est adapté du (bientôt obsolète) Ipchains-HOWTO :

Il y a quatre bits rarement utilisés dans l'en-tête IP, appelés bits de Type de Service (TOS). Ils affectent la manière dont les paquets sont traités. Les quatre bits sont "Délai Minimum", "Débit Maximum", "Fiabilité Maximum" et "Coût Minimum". Seul un de ces bits peut être positionné. Rob van Nieuwkerk, l'auteur du code TOS-mangling dans ipchains, le configure comme suit :

```
Le "Délai Minimum" est particulièrement important pour moi. Je le positionne à 1 pour
les paquets interactifs sur mon routeur (Linux) qui envoie le trafic vers l'extérieur.
Je suis derrière un modem à 33,6 Kbps. Linux répartit les paquets dans trois files
d'attente. De cette manière, j'obtiens des performances acceptables pour le trafic
interactif tout en téléchargeant en même temps.
```

L'utilisation la plus commune est de configurer les connections telnet et ftp à "Délai Minimum" et les données FTP à "Débit Maximum". Cela serait fait comme suit, sur mon routeur :

```
# iptables -A PREROUTING -t mangle -p tcp --sport telnet \
  -j TOS --set-tos Minimize-Delay
# iptables -A PREROUTING -t mangle -p tcp --sport ftp \
  -j TOS --set-tos Minimize-Delay
# iptables -A PREROUTING -t mangle -p tcp --sport ftp-data \
  -j TOS --set-tos Maximize-Throughput
```

En fait, cela ne marche que pour les données venant d'un telnet extérieur vers votre ordinateur local. Dans l'autre sens, ça se fait tout seul : telnet, ssh, et consorts configurent le champ TOS automatiquement pour les paquets sortants.

Si vous avez un client incapable de le faire, vous pouvez toujours le faire avec netfilter. Sur votre machine locale :

```
# iptables -A OUTPUT -t mangle -p tcp --dport telnet \
  -j TOS --set-tos Minimize-Delay
# iptables -A OUTPUT -t mangle -p tcp --dport ftp \
  -j TOS --set-tos Minimize-Delay
# iptables -A OUTPUT -t mangle -p tcp --dport ftp-data \
  -j TOS --set-tos Maximize-Throughput
```

15 Routage avancé sur Linux

Cette section est destinée à tous les gens qui voudraient savoir comment fonctionne le système complet ou qui ont une configuration si bizarre qu'ils ont besoin d'informations "bas niveau" pour le faire fonctionner.

Cette section est complètement facultative. Il est tout à fait possible qu'elle soit trop complexe et pas vraiment destinée aux utilisateurs normaux. Vous avez été averti.

FIXME : Décider ce qui doit réellement être ici.

15.1 Comment la mise en file d'attente des paquets fonctionne-t-elle en pratique ?

C'est la base du fonctionnement du système de mise en file d'attente d'un paquet.

Lister les étapes que le noyau réalise pour classifier un paquet, etc.

FIXME : à écrire.

15.2 Utilisation avancée du système de mise en file d'attente des paquets

Passer par l'exemple extrêmement rusé de Alexeys impliquant les bits inutilisés dans le champs TOS.

FIXME : à écrire.

15.3 D'autres systèmes de mise en forme des paquets

Je voudrais inclure une brève description sur les autres systèmes de mise en forme des paquets dans d'autres systèmes d'exploitation et les comparer à celui de Linux. Puisque Linux est l'un des rares OS qui ont une pile TCP/IP originale (non dérivée de BSD), je pense qu'il serait intéressant de savoir comment les autres font cela.

Malheureusement je n'ai aucune expérience sur d'autres systèmes, aussi je ne peux pas l'écrire.

FIXME : Personne ? - Martijn

16 Routage Dynamique - OSPF et BGP

Si votre réseau commence à devenir vraiment gros, ou si vous commencez à considérer Internet comme votre propre réseau, vous avez besoin d'outils qui routent dynamiquement vos données. Les sites sont souvent reliés entre eux par de multiples liens, et de nouveaux liens surgissent en permanence.

L'Internet utilise la plupart du temps les standards OSPF et BGP4 (rfc1771). Linux supporte les deux, par le biais de `gated` et `zebra`.

Ce sujet est pour le moment hors du propos de ce document, mais laissez-nous vous diriger vers des travaux de référence :

Vue d'ensemble :

Cisco Systems *Cisco Systems Designing large-scale IP internetworks* `<http://www.cisco.com/univercd/cc/td/doc/cisintwk/idg4/nd2003.htm>`

Pour OSPF :

Moy, John T. "OSPF. The anatomy of an Internet routing protocol" Addison Wesley. Reading, MA. 1998.

Halabi a aussi écrit un très bon guide sur la conception du routage OSPF, mais il semble avoir été effacé du site Web de Cisco.

Pour BGP :

Halabi, Bassam "Internet routing architectures" Cisco Press (New Riders Publishing). Indianapolis, IN. 1997.

Il existe aussi

Cisco Systems

Using the Border Gateway Protocol for Interdomain Routing `<http://www.cisco.com/univercd/cc/td/doc/cisintwk/ics/icsbgp4.htm>`

Bien que les exemples soient spécifiques à Cisco, ils sont remarquablement semblables au langage de configuration de Zebra :-)

17 Lectures supplémentaires

http ://snafu.freedom.org/linux2.2/iproute-notes.html <http://snafu.freedom.org/linux2.2/iproute-notes.html>

Contient beaucoup d'informations techniques, et de commentaires sur le noyau.

http ://www.davin.ottawa.on.ca/ols/ <http://www.davin.ottawa.on.ca/ols/>

Transparents de Jamal Hadi, un des auteurs du contrôleur de trafic de Linux.

http ://defiant.coinet.com/iproute2/ip-cref/ <http://defiant.coinet.com/iproute2/ip-cref/>

Version HTML de la documentation LaTeX d'Alexeys ; explique une partie d'iproute2 en détails.

http ://www.aciri.org/floyd/cbq.html <http://www.aciri.org/floyd/cbq.html>

Sally Floyd a une bonne page sur CBQ, incluant ses publications originales. Aucune n'est spécifique à Linux, mais il y a un travail de discussion sur la théorie et l'utilisation de CBQ. Contenu très technique, mais une bonne lecture pour ceux qui sont intéressés.

http ://ceti.pl/˜ekravietz/cbq/NET4_tc.html <http://ceti.pl/˜ekravietz/cbq/NET4_tc.html>

Un autre HOWTO, en polonais ! Vous pouvez cependant copier/coller les lignes de commandes, elles fonctionnent de la même façon dans toutes les langues. L'auteur travaille en collaboration avec nous et sera peut être bientôt un auteur de sections de cet HOWTO.

Les services différenciés sur Linux <http://snafu.freedom.org/linux2.2/docs/draft-almesberger-wajhak-diffserv-linux-00.txt>

Discussion sur l'utilisation de Linux dans un environnement compatible diffserv. Assez loin des besoins quotidiens de votre routeur, mais néanmoins très intéressant. Nous pourrons inclure une section sur le sujet, plus tard.

IOS Committed Access Rate <http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/car.htm>

Des gens de Cisco qui ont pris la louable habitude de mettre leur documentation en ligne. La syntaxe de Cisco est différente mais les concepts sont identiques, sauf qu'on fait mieux, et sans matériel qui coûte autant qu'une voiture :-)

TCP/IP Illustrated, volume 1, W. Richard Stevens, ISBN 0-201-63346-9

Sa lecture est indispensable si vous voulez réellement comprendre TCP/IP, et de plus elle est divertissante.

18 Remerciements

Notre but est de faire la liste de tous les personnes qui ont contribué à ce HOWTO, ou qui nous ont aidés à expliquer le fonctionnement des choses.

Alors qu'il n'existe pas de liste dans Netfilter, nous souhaitons saluer les personnes qui ont apporté leur aide.

- Jamal Hadi <hadi%cyberus.ca>
- Nadeem Hasan <nhasan@usa.net>
- Jason Lunz <j@cc.gatech.edu>
- Alexey Mahotkin <alexm@formulabez.ru>
- Pawel Krawczyk <kravietz%alfa.ceti.pl>
- Wim van der Most
- Glen Turner <glen.turner%aarnet.edu.au>
- Song Wang <wsong@ece.uci.edu>