Routing avanzado con el núcleo Linux

Eric Van Buggenhaut

Andago

eric@andago.com

El núcleo Linux, a partir de la versión 2.4 nos ofrece una interfaz que permite implementar herramientas profesionales de alto nivel en cuanto a la gestión del tráfico de paquetes IP. Nos permite hacer cosas como túneles IP, tablas de routing múltiples, reserva de ancho de banda, multicasting, proxy ARP y mucho más. Esas funcionalidades estaban hasta ahora solo disponibles en routers propietarios de gama alta y de precio casi prohibitivo. El núcleo linux nos permite implementarlas de modo más seguro, mas económico y con más rendimiento, además de desarrollar nuestras propias herramientas específicas.

1. Introducción

A partir de la versión 2.4 del núcleo Linux, está disponible un socket llamado NETLINK que permite implementar en espacio usuario (user space) código de gestión de paquetes y tráfico IP. Teóricamente, este socket, por su naturaleza, nos permite implementar cualquier código.

Sin embargo, nos vamos a centrar en el código escrito por Alexey Kuznetsov y disponible bajo el nombre 'iproute' bajo licencia GPL. Está accesible en ftp://ftp.inr.ac.ru/ip-routing

El código está disponible como paquete Debian iproute*deb

Vamos a ver a lo largo de esta charla algunas de las cosas que nos permite hacer este código:

- unificar los comandos relacionados con la gestión del tráfico IP, sea de redes, de interfaces, ...
- monitorización de los periféricos, direcciones y rutas.
- gestión de tablas ARP
- uso de tablas de routing múltiples
- creación de túneles IP
- reserva de ancho de banda

2. Configurar el sistema

Para que funcione el iproute, necesitamos configurar el núcleo para que provea el socket NETLINK que nos interesa. En /usr/src/linux/.config vienen bastante opciones que nos permiten adaptar el kernel a nuestras necesidades:

```
CONFIG_NETLINK=y
CONFIG_RTNETLINK=y
# CONFIG_NETLINK_DEV is not set
CONFIG_NETFILTER=y
CONFIG_NETFILTER_DEBUG=y
# CONFIG_FILTER is not set
CONFIG_UNIX=y
CONFIG_INET=y
# CONFIG_IP_MULTICAST is not set
CONFIG_IP_ADVANCED_ROUTER=y
CONFIG_RTNETLINK=y
CONFIG_NETLINK=y
CONFIG_IP_MULTIPLE_TABLES=y
CONFIG_IP_ROUTE_FWMARK=y
CONFIG_IP_ROUTE_NAT=y
CONFIG_IP_ROUTE_MULTIPATH=y
CONFIG_IP_ROUTE_TOS=y
CONFIG_IP_ROUTE_VERBOSE=y
CONFIG_IP_ROUTE_LARGE_TABLES=y
# CONFIG_IP_PNP is not set
CONFIG_NET_IPIP=m
CONFIG_NET_IPGRE=m
# CONFIG_ARPD is not set
# CONFIG_INET_ECN is not set
```

3. El comando ip

Actualmente usamos varios comandos para gestionar el tráfico IP y todo lo que le rodea: interfaces, rutas, túneles, ... Se decidió unificar el conjunto y proveer así un sólo comando con una sintaxis coherente y global. El comando se llama ’ip’ y tiene la siguiente sintaxis:

```
ip [OPTIONS] OBJECT [COMMAND [ARGUMENTS]]
```

3.1. OPTIONS

Son varias opciones que influyen el comportamiento general de la herramienta. Todas las opciones empiezan por el carácter ’-’ y se pueden abreviar. Algunas opciones son:

- -s, -stats, -statistics obtener más información
- -f, -family especifica que familia de protocolo usar: inet, inet6 o link.
- -r, -resolve imprime nombres DNS en lugar de direcciones de host

3.2. OBJECT

Es el objeto que queremos manejar o del cual buscamos informaciones. He aquí algunos ejemplos que también pueden ser abreviados:

- link,l -- periférico de red
- address,a -- dirección (IPv4 o IPv6) de periférico
- route,r -- entrada en la tabla de routing
- rule,ru -- regla en la base de datos de política (policy database)
- maddress,maddr -- dirección multicast
- tunnel,t -- tunnel sobre IP

3.3. COMMAND

Es el comando que se aplica al objeto. Se puede abreviar también:

- add,a -- añadir un objeto
- del,d -- borrar un objeto
- set,s -- ajustar un objeto
- show,list,l -- ver un objeto

3.4. Mensajes de error

Algunos posibles errores son:

- Wrong syntax of command line -- problema de sintaxis
- el núcleo devuelve un error a una petición NETLINK -- En este caso, ip imprime el mensaje de error prefijado por "RTNETLINK answers:"
- Cannot open netlink socket: Invalid value -- Netlink no está configurado en el núcleo
- Cannot talk to rtnetlink: Connection refused -- RTNETLINK no está configurado en el núcleo
- Cannot send dumb request: Connection refused -- RTNETLINK no está configurado en el núcleo
- RTNETLINK error: Invalid argument -- CONFIG_IP_MULTIPLES_TABLES no está configurado en el núcleo

4. ip link -- manejar las interfaces

4.1. ip link set -- cambiar los atributos de la interfaz

- dev NAME: especifica de que interfaz se trata
- up/down: cambiar el estado de la interfaz
- name NAME: cambiar nombre de la interfaz
- mtu NUMBER: cambiar MTU de la interfaz
- ejemplo:

```
ip link set dummy up
```

4.2. ip link show -- ver los atributos

- dev NAME: mostrar la interfaz especificada
- up: mostrar solo las interfaces ’up’
- ejemplos:

```
[eric@mrmime:~]$ ip l l
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc
                                    pfifo_fast qlen 100
link/ether 00:05:1c:01:b1:33 brd ff:ff:ff:ff:ff:ff
[eric@mrmime:~]$
```

La primera linea de cada entrada da un número único a la interfaz, su nombre (que puede ser cambiado), así como varias informaciones sobre el estado de la interfaz. La segunda línea da informaciones sobre el tipo de interfaz de que se trata, la dirección de la interfaz a nivel de la capa ’layer’ (en el caso de ethernet, la dirección MAC).

La opción -s nos permite ver estadísticas de la interfaz:

```
[eric@mrmime:~]$ ip -s l l
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  RX: bytes  packets  errors  dropped overrun mcast
  1368991    5872     0       0       0       0
  TX: bytes  packets  errors  dropped carrier collsns
  1368991    5872     0       0       0       0
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc
    pfifo_fast qlen 100
  link/ether 00:05:1c:01:b1:33 brd ff:ff:ff:ff:ff:ff
  RX: bytes  packets  errors  dropped overrun mcast
  1789685066 1975826  99656   0       0       0
  TX: bytes  packets  errors  dropped carrier collsns
  1619835989 2304565  69      16      4       1762589
[eric@mrmime:~]$
```

Los parámetros son similares al antiguo comando ’ifconfig’.

4.3. ip address -- gestión de las direcciones de interfaz

ip addr permite ver las direcciones de interfaz, añadir nuevas direcciones o borrarlas. Es importante destacar que a partir de iproute, las interfaces físicas y las direcciones son totalmente disociadas, eso significa que una interfaz puede tener varias direcciones sin necesitad de crear un alias como ocurría en el caso anterior.

- ip addr show: ver direcciones de protocolo

```
mrmime:~# ip a l
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc
      pfifo_fast qlen 100
    link/ether 00:05:1c:01:b1:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.71/24 brd 192.168.2.255 scope global eth0
mrmime:~#
```

- ip addr add: añadir nueva dirección

```
mrmime:~# ip a l
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc
      pfifo_fast qlen 100
    link/ether 00:05:1c:01:b1:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.71/24 brd 192.168.2.255 scope global eth0
mrmime:~# ip a a 10.0.0.1 dev eth0
mrmime:~# ip a l
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc
      pfifo_fast qlen 100
    link/ether 00:05:1c:01:b1:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.71/24 brd 192.168.2.255 scope global eth0
    inet 10.0.0.1/32 scope global eth0
mrmime:~#
```

- ip addr del: borrar una dirección

```
mrmime:~# ip a d 10.0.0.1 dev eth0
mrmime:~# ip a l
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc
     pfifo_fast qlen 100
    link/ether 00:05:1c:01:b1:33 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.71/24 brd 192.168.2.255 scope global eth0
mrmime:~#
```

5. Gestión de tablas ARP

Las tablas ARP establecen enlaces entre las capas de protocolo y de ’link’ (en caso de una red local Ethernet, sería entre dirección IP y dirección MAC). Cada host de una subred necesita conocer la dirección física de los demás para poder mandar paquetes a los destinatarios adecuados. Estas direcciones se almacenan en un caché ARP.

En caso de no conocer la dicha dirección MAC, se hace una petición de broadcast (por ejemplo cuando se enciende el equipo) que sería cómo "Hola ! Quién es la dirección IP w.x.y.z ?". La máquina con tal dirección IP contesta : "Yo soy la máquina w.x.y.z y mi MAC es 00:ad:f3:b1:22:4e". Entonces, nuestra máquina almacena esta MAC en su caché.

El Objeto ’neighbour’ del comando ip gestiona el cache ARP. Estos son algunos ejemplos:

```
mrmime:~# ip neigh ls
192.168.2.5 dev eth0 lladdr 00:c0:ca:15:80:9c nud reachable
192.168.2.34 dev eth0 lladdr 00:05:1c:01:9d:c3 nud delay
192.168.2.72 dev eth0 lladdr 00:05:1c:01:6c:a9 nud reachable
192.168.2.1 dev eth0 lladdr 00:c0:ca:15:81:07 nud reachable
192.168.2.3 dev eth0 lladdr 00:01:02:ad:08:da nud stale
192.168.2.70 dev eth0 lladdr 00:40:f6:2c:27:13 nud reachable
192.168.2.21 dev eth0 lladdr 00:50:fc:42:07:b4 nud reachable
192.168.2.52 dev eth0 lladdr 00:05:1c:01:5e:1b nud delay
mrmime:~#
```

Puedo borrar una entrada:

```
mrmime:~# ip n d 192.168.2.52 dev eth0
mrmime:~# ip n l
192.168.2.5 dev eth0 lladdr 00:c0:ca:15:80:9c nud reachable
192.168.2.34 dev eth0 lladdr 00:05:1c:01:9d:c3 nud reachable
192.168.2.72 dev eth0 lladdr 00:05:1c:01:6c:a9 nud reachable
192.168.2.1 dev eth0  nud failed
192.168.2.3 dev eth0 lladdr 00:01:02:ad:08:da nud reachable
192.168.2.70 dev eth0 lladdr 00:40:f6:2c:27:13 nud reachable
192.168.2.21 dev eth0 lladdr 00:50:fc:42:07:b4 nud reachable
192.168.2.52 dev eth0 lladdr 00:05:1c:01:5e:1b nud reachable
```

Que paso ? Realmente la entrada correspondiente no desaparecerá de inmediato. Se quedara hasta que el último cliente que la usa la ’libere’

Los otros comandos para añadir o cambiar una entrada son:

- ip neigh add
- ip neigh change

6. Tablas de routing múltiples

ip nos permite trabajar con tablas de routing múltiples, una gran novedad en la gestión de tráfico IP. ¨De qué se trata ? Hasta ahora habíamos tenido, por cada sistema, una tabla de routing única, que define a qué interfaz esta conectada una red, donde se encuentra el gateway, etc ... Gracias a iproute, podemos trabajar con varias tablas de routing a la vez y elegir que tabla usar según las características del paquete IP.

Imaginemos por ejemplo que tenemos un router con dos interfaces de conexión a Internet. Una interfaz RDSI más lenta pero barata y una interfaz ADSL rápida, pero más cara. La ventaja es que se puede decidir, gracias a iproute, que interfaz será usada, según los paquetes que hay que mandar. Por ejemplo podemos decidir que los paquetes SMTP saldrán por la interfaz RDSI lenta (no hay prisa). A revés, si tenemos aplicaciones de videoconferencia, queremos que vayan por la interfaz ADSL rápida. En este caso, la política de routing se basa en el puerto de destino de los paquetes IP.

También podemos imaginar basar la política de routing sobre la dirección IP de origen de los paquetes. Sería el caso si queremos dar prioridad a determinados servicios de una empresa. Por ejemplo, la dirección de la empresa usará la conexión ADSL rápida, mientras el servicio de mecanografía vera su tráfico dirigido por la interfaz lenta.

En realidad, podemos basarnos en muchos parámetros para establecer nuestra política de routing: IP de origen, IP de destino, puerto de origen, puerto de destino, protocolo usado, TOS e interfaz de llegada.

Hay que destacar que para hacer comprobaciones sobre protocolos IP y puertos de transporte, hay que usar el sistema conjuntamente con ipchains que nos provee fwmark, un sistema para marcar paquetes, los cuales veremos más adelante. Por defecto, hay 3 tablas de routing en la base de datos 'routing policy database':

```
[eric@mrmime:~]$ ip ru l
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
[eric@mrmime:~]$
```

El antiguo comando 'route' nos enseña la tabla 'main'. Las dos otras son nuevas.

La tabla 'local' es especial, no puede ser borrada y tiene la prioridad más alta (0). Se usa para direcciones locales y de broadcast.

La tabla 'main' es la tabla clásica que nos devuelve el antiguo comando 'route'. Se puede borrar o cambiar.

La tabla 'default' esta vacía y se reserva para procesos de post-routing (si las reglas anteriores no coinciden). También se puede borrar.

```
[eric@mrmime:~]$ ip r l 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
 192.168.2.0/24 dev eth0  proto kernel  scope link
                src 192.168.2.71
 127.0.0.0/8 dev lo  scope link
 default via 212.64.94.1 dev ppp0
[eric@mrmime:~]$
```

6.1. Ejemplo sencillo de routing según origen

Imaginamos que las dos interfaces de red que hemos mencionado antes son: 212.64.94.1 para el tráfico de alta velocidad 195.96.98.253 para el tráfico lento.

Pedro no se ha portado bien esta semana así que le vamos a redireccionar a través de la conexión lenta. Generamos una regla que se llama ’Pedro’:

```
# echo 200 Pedro >> /etc/iproute2/rt_tables
# ip rule add from 192.168.2.35 table Pedro
# ip rule ls
0:      from all lookup local
32765:  from 192.168.2.35 lookup Pedro
32766:  from all lookup main
32767:  from all lookup default
```

Esta regla especifica que todo el tráfico que viene con IP de origen 192.168.2.35 usa la tabla de routing llamada Pedro. Necesitamos entonces crear la misma tabla Pedro:

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

Así de simple.

A continuación vemos los identificadores que se pueden usar en la base de datos:

- from: determina la origen del paquete
- to: determina el destino del paquete
- iis: determina la interfaz de llegada
- tos: determina el valor de TOS
- fwmark: determina el valor de ’marca’ del paquete (puesto por iptables por ej., ver el ejemplo siguiente)

Es importante no confundir ’tablas de routing’ y ’reglas’. Las reglas apuntan a tablas de routing, varias reglas pueden apuntar a la misma tabla de routing. De la misma manera, una tabla de routing puede no tener ninguna regla apuntando a ella sin dejar de existir.

6.2. Ejemplo más complejo con marcación de paquetes

En el ejemplo previo, hemos dirigido nuestros paquetes según su dirección IP de origen. Si queremos, por otra parte, basar nuestra política sobre el tipo de tráfico (correo, web, vídeo), usaremos la herramienta de marcación de paquetes proveída por netfilter (iptables).

Siguiendo con los datos previos, vemos ahora como desviar nuestro tráfico de web saliente hacia la interfaz rápida (212.64.94.1, ver arriba).

Marcamos los paquetes que tienen como destino el puerto 25 (SMTP) con un número '1':

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

Ahora dirigimos los paquetes marcados '1' a una tabla de routing especifica, en este caso, la llamamos web.out:

```
# echo 201 mail.out >> /etc/iproute2/rt_tables
# ip rule add fwmark 1 table web.out
# ip rule ls
0:      from all lookup local
32764:  from all fwmark 1 lookup web.out
32765:  from 192.168.2.35 lookup Pedro
32766:  from all lookup main
32767:  from all lookup default
```

Sólo nos hace falta ahora crear la tabla de routing:

```
#ip route add default via 195.96.98.253 dev ppp0 table web.out
```

Ya lo tenemos todo. La regla de marca que usamos es muy simple, sólo se basa en el puerto de destino del paquete. Obviamente, con la flexibilidad que nos ofrece iptables, podemos hacerlo mucho más complejo, añadiendo excepciones, etc.

7. ip tunnel -- Túneles

Sin necesidades de usar herramientas externas y complicadas como PPTP, iptunnel, etc. Podemos crear túneles cifrados o no, con el mismo comando ip. Existen tres tipos de túneles disponibles: IPIP (IP sobre IP), sit y GRE (el protocolo desarrollado por Cisco). Se trata realmente de encapsular paquetes IP en otros paquetes IPv4 y mandarlos a través de una infraestructura IP.

7.1. ip tun [add|change|delete] -- 'abrir/cambiar/cerrar' un túnel

Los argumentos posibles son:

- name NOMBRE -- selecciona el nombre del túnel
- mode MODO -- hay 3 modos disponibles: ipip, sit y gre

El modo ipip corresponde a un simple túnel IP sobre IP. Se encapsulan los paquetes sin más. El modo sit se usa para túneles IPv6. El modo gre corresponde a los túneles GRE especificados por la compañía Cisco, que son túneles IP sobre IP cifrados.

- remote DIRECCION -- dirección de 'salida' del tunel
- local DIRECCION -- dirección local de 'entrada' del tunel
- dev PERIFERICO-- nombre del periférico a través del que se envían los paquetes

Por ejemplo, para crear un tunel IPv6 sobre IPv4,

```
ip tunl add MiTunel mode sit remote 192.31.7.104\
local 192.203.80.142
```

7.2. ip tun show -- ver los túneles

```
mrmime:~# ip tu ls
tunl0: ip/ip  remote any  local any  ttl inherit  nopmtudisc
mitun: ip/ip  remote 192.168.2.5  local 192.168.2.71  ttl inherit
mrmime:~#
```

o con estadísticas

```
mrmime:~# ip -s tu ls mitun
mitun: ip/ip  remote 192.168.2.5  local 192.168.2.71  ttl inherit
RX: Packets    Bytes        Errors CsumErrs OutOfSeq Mcasts
    0          0            0      0        0        0
TX: Packets    Bytes        Errors DeadLoop NoRoute  NoBufs
    0          0            0      0        0        0
mrmime:~#
```

8. Reserva de ancho de banda

Una de las grandes ventajas de Linux es que va mucho más allá de lo que existe en materia de sistema de gestión de ancho de banda de tipo propietario, que cuestan cientos de miles de pesetas.

Para el controlo de tráfico, se usan dos conceptos: filtros y colas. Los filtros ponen los paquetes en las colas y las colas deciden qué paquetes mandar antes que otros. Luego veremos más en detalle la relación entre colas y ancho de banda.

Hay muchos tipos de filtros distintos, los más comunes son ’fwmark’ y ’u32’. El primero nos deja usar el código de netfilter (iptables) para seleccionar tráfico, mientras que el segundo nos permite seleccionar el tráfico basándose en cualquier cabecera de paquete.

Vamos a demostrarlo con un ejemplo ficticio de ISP que necesita hacer reparto de ancho de banda entre sus clientes.

8.1. Qué son colas ?

Las colas determinan el orden en que se mandan los paquetes. Qué tiene que ver con nuestro ancho de banda ?

Imaginamos una caja de supermercado donde la gente hace cola para pagar sus compras. La ultima persona llegada se pone al final de la cola y espera su turno: es una cola FIFO (First In, First Out). Ahora si dejamos ciertas personas siempre ponerse en medio de la cola, serán atendidas antes y podrán comprar más rápidamente.

Internet esta basado principalmente en TCP/IP y TCP/IP no sabe nada de ancho de banda. Lo que hace una máquina es mandar datos cada vez más rápido hasta que se detecte que algunos paquetes se están perdiendo, luego ralentiza el proceso.

Es el equivalente de no leer la mitad de los emails recibidos, esperando que la gente deje de mandar correo. La diferencia es que con Internet sí funciona.

8.2. Nuestro ISP

Nuestro router Linux tiene 2 interfaces eth0 y eth1. eth0 esta conectado a nuestra red de clientes y eth1 es nuestra conexión al backbone Internet.

Sólo podemos limitar lo que mandamos así que necesitamos dos tablas de reglas. Modificaremos la cola de eth0 para decidir la velocidad a la que se mandan los datos a nuestros clientes, su ancho de banda o su 'download speed'.

Luego modificamos eth1 para especificar la velocidad de envío de datos a Internet. Es lo que se denomina 'upload speed' para nuestros clientes.

8.3. Usando CBQ

Con CBQ podemos usar clases de usuarios (así como subclases). Vamos a crear dos: 'ISP' para nuestros clientes y 'Office' para la red corporativa. Disponemos de 10 Mbit de ancho de banda, vamos a dar 8 a ISP y 2 a Office.

La herramienta que usamos es 'tc'

Definimos la regla de cola (qdisc: queueing discipline) para eth0. Con 'root' indicamos cuál es la disciplina raíz. El 'handle' es el identificador de la regla. Luego indicamos al núcleo que tiene 10Mbit disponibles y que el tamaño medio de los paquetes es más o menos 1000 octetos:

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

Ahora generamos nuestra clase raíz de la que cuelgan las demás. 'parent 10:0' indica que desciende del 'handle' '10:'. Le asignamos el identificador '10:1'. Se especifica también el MTU (1514). Para información, el MTU (Maximum Transmission Unit) es el tamaño máximo posible de datos transportados por un 'frame' (por ejemplo un datagramo IP). En el caso de una conexion ethernet, el MTU es de 1500:

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

Generamos luego nuestra clase ISP: le dedicamos 8 Mbit e indicamos que no se puede superar, con ’bounded’, si no, iría restando ancho de banda de otras clases:

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

Este es el comando que genera la clase Office:

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

Hasta ahora hemos indicado al núcleo cuáles son nuestras clases pero no cómo gestionar las colas. El procedimiento es el siguiente:

```
# 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
```

En este caso, usamos la regla ’Stochastic Fairness Queueing’ (sfq), que no es totalmente imparcial, pero que funciona bien para un ancho de banda bastante elevado, sin cargar demasiado el núcleo.

Hay otras reglas como ’Token Bucket Filter’ pero consumen mas ciclos de CPU.

Sólo nos queda hacer una cosa: indicar al núcleo qué paquetes pertenecen a qué clase:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 100 \
  u32 match ip \
150.151.23.24 flowid 10:200
# tc filter add dev eth0 parent 10:0 protocol ip prio 25 \
  u32 match ip \
150.151.0.0/16 flowid 10:10
```

Aquí consideramos que Office solo tiene 1 dirección IP para todo el tráfico y que las demás son de ISP.

Esta parte servía para dividir el tráfico ’downstream’. Hay que hacer lo mismo con el tráfico saliente:

```
# tc qdisc add dev eth1 root handle 20: cbq bandwidth \
  10Mbit avpkt 10
# tc class add dev eth1 parent 20:0 classid 20:1 cbq \
bandwidth 10Mbit 10Mbit allot 1514 weight 1Mbit prio 8 \
```

```
maxburst 20 avpkt 1000
# tc class add dev eth1 parent 20:1 classid 20:100 cbq \
bandwidth 10Mb 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 10Mb 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 150.151.23.24 flowid 20:200
# tc filter add dev eth1 parent 20:0 protocol ip prio 25 \
u32 match ip 150.151.0.0/16 flowid 20:100
```

Pues muy bien, tenemos nuestro sistema de ancho de banda funcionando !

8.4. Más disciplinas de cola

Hay muchas disciplinas de cola bajo Linux. Cada una tiene sus ventajas y inconvenientes.

8.4.1. pfifo_fast

Desde luego es la cola más usada. Como indica su nombre, es una cola FIFO y entonces los paquetes no reciben tratamiento especial.

8.4.2. Stochastic Fairness Queue (sfq)

Es el tipo de cola que hemos visto antes. No es muy determinista pero trabaja bastante bien. Su principal ventaja es que consume poco ciclos de CPU y poca memoria. Para tener un sistema de cola totalmente imparcial, habría que grabar todos los paquetes que pasan por la máquina.

Existen varios algoritmos ’imparciales’ (fair). SFQ es uno de ellos, y se basa en lo que llamaremos conversaciones. Una conversación es una secuencia de paquetes de datos que tienen suficientes parámetros como para distinguirlos de las otras conversaciones. En este caso, los parámetros usados son la dirección de origen y de destino de los paquetes, así cómo el protocolo usado.

Se otorgan colas FIFO dinámicamente a las conversaciones que surgen, una cola por conversación. La disciplina de envío es de tipo round-robin, es decir se envía un paquete de cada cola secuencialmente. Un inconveniente de SFQ es que no puede distinguir tráfico interactivo de tráfico de masa. Por esta razón se suele usar CBQ antes de SFQ

8.4.3. Token Bucket Filter

TBF es un buffer (bucket/cubo) rellenado constantemente por trozos virtuales de información (tokens) a velocidad especifica (token rate). El parámetro realmente importante es el tamaño del cubo.

Cada token que llega permite a un paquete llegando a la cola ser enviado y luego se destruye. Tenemos dos flujos que pueden dar lugar a tres posibilidades:

- Si los datos llegan al TBF a la misma velocidad que los tokens, cada paquete empareja un token y pasa la cola sin demorar.
- Si los paquetes de datos llegan menos rápido que los tokens, los tokens se van acumulando hasta rellenar el cubo. Si llegan todavía más tokens, se pierden. Los tokens que se han ido acumulando pueden ser usado luego para amortiguar picos de tráfico.
- Si los paquetes de datos llegan más rápido que los tokens, cuando no hay más tokens, se tiran los paquetes que desbordan

La última posibilidad es importante, en el sentido que permite definir el ancho de banda disponible para los datos. La acumulación de tokens permite amortiguar unos picos de tráfico pero no una sobrecarga continúa.

8.4.4. Random Early Detect

RED es un gestor de cola ’inteligente’. Cuando se inicia una sesión TCP/IP, ninguno de los dos hosts sabe cual es el ancho de banda disponible. Así que TCP/IP empieza a transmitir paquetes cada vez más rápido. Cuando se esta ocupando todo el ancho de banda, se desechan los paquetes (ver arriba).

Lo que hace RED es que finge una congestión antes de que el ancho de banda sea realmente enteramente ocupado. Es una técnica muy útil para tráfico de ancho de banda muy grande. RED se usa principalmente en los routers de Backbone, es decir donde el ancho de banda supera 100 Mbits/s. En este caso no se puede hacer un reparto de ancho de banda ’imparcial’. Más bien se hace un reparto de tipo estadística.

8.4.5. Ingress policer qdisc

Este tipo de cola es interesante si se necesita limitar el ancho de banda de una máquina sin ayuda de una caja externa. Se puede determinar el ancho de banda internamente y decidir tirar paquetes cuando se desborda.

En breve, es un sistema práctico cuando uno quiere limitar su velocidad de baja de ficheros, para dejar un poco de ancho de banda a sus colegas.

8.4.6. DSMARK

Para acabar este serie, hay que mencionar Dsmark también. Es probablemente el gestor de cola más complicado que existe. Su explicación sola necesitaría seguramente una buena así que no vamos a entrar en los detalles.

Dsmark es una disciplina de cola que permite ofrecer Servicios Diferenciados (Differentiated Services, o DiffServ o DS). Hoy en día conocemos dos arquitecturas de QoS (Quality of Service, Calidad de Servicio).

Una de los primeros intentos para ofrecer QoS en IP fue de usar el campo TOS (Type of Service) en las cabeceras IP. Cambiando su valor, podíamos elegir un nivel de débito o de fiabilidad. Pero este método no daba suficiente flexibilidad. Una de la nuevas arquitecturas que apareció fue DiffServ que usa el campo DS de las cabeceras IP.

Para más informaciones os aconsejo dirigiros a la página
http://www.ietf.org/html.carters/diffserv-charter.html

8.5. Para concluir

Esperamos que esta breve introducción fue interesante y os presento un poco todas las posibilidades del núcleo Linux a nivel de gestión avanzada de tráfico. Hay muchos campos que no hemos podido mencionar debido a la complejidad del asunto.

También hay que destacar que es un dominio en pleno desarrollo y evolución. Habría que mencionar los clasificadores de cola, todos los parámetros de configuración del núcleo a nivel TCP/IP, el bridging y el proxy-arp, el routing dinámico, limitación del tráfico ICMP, etc, etc.

Bibliografía

S. Alexey N. Kuznetsov, *IP Command Reference: http://snafu.freedom.org/linux2.2/docs/ip-cref/ip-cref.html* , Descripción de la herramienta ip del paquete iproute2 .

Bert Hubert et al., *Linux 2.4 Advanced Routing HOWTO: http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html (http://www.linuxdoc.org/HOWTO/Adv-Routing-HOWTO.html)* , Guía COMO de routing avanzado .