CONTROL DE TRÁFICO UTILIZANDO LINUX

ÍNDICE

CONTROL DE TRÁFICO UTILIZANDO LINUX

0. INTRODUCCIÓN

El objeto de este trabajo es describir las posibilidades que ofrece el Sistema Operativo Linux dentro del ámbito de las redes de comunicaciones.
Comenzaremos haciendo una breve descripción del potencial de Linux en el campo del networking a través de su herramienta IPROUTE2. Sin embargo, el grueso de nuestro trabajo intentará recoger de forma sintética pero lo más completa posible todas las posibilidades que ofrece Linux en el ámbito del Control de Tráfico. Para ello estudiaremos toda la teoría de disciplinas de colas implementada por Linux, acompañada con ejemplos puntuales.
Por último desarrollaremos un caso real en el que se ha aplicado de forma directa parte de ese potencial que Linux pone a nuestra disposición.

1. POSIBILIDADES DE LINUX DENTRO DEL ÁMBITO DEL NETWORKING

A partir de las versiones del kernel 2.2.X, y hasta la 2.4.X, los avances de Linux dentro del ámbito del networking han sido espectaculares.
Actualmente con Linux podemos implementar facilidades como:

- Regular a nuestro antojo (abriendo o cerrando) el ancho de banda de un interfaz de red.
- Repartir, en función de multitud de criterios, el ancho de banda de nuestro interfaz de red.
- Proteger nuestra red de ataques tan clásicos como el Deny of Service.
- Proteger a Internet de nuestros propios clientes.
- Multiplexar varios servidores como uno solo, permitiendo implementar facilidades como el balanceo de carga o la alta disponibilidad.
- Hacer el enrutado según criterios tan variados como el usuario, dirección MAC, dirección IP de origen, tipo de servicio, hora del día, etc.

Pese a que todas estas facilidades están disponibles, es cierto que no se está haciendo un gran uso de ellas. La razón básica para esto hay que buscarla en la escasa documentación y soporte disponible. Como veremos más adelante, gran parte del trabajo con estas facilidades de Linux requiere mucho esfuerzo en pruebas e investigación.

2. INTRODUCCIÓN A IPROUTE2

El kernel de Linux a partir de la versión 2.2.X posee un subsistema de red totalmente rediseñado. La principal razón para ello fue que ante la necesidad de nuevas facilidades, que han ido surgiendo dentro del ámbito del networking, Linux fue añadiendo parches a su implementación de la pila de protocolos TCP/IP, lo cual provocó que el nivel de complejidad creciera enormemente haciéndolo prácticamente inmanejable.

A partir del kernel 2.2.X, y gracias a este nuevo diseño, facilidades como el routing, corta-fuegos y código de clasificación de tráfico, son mucho más potentes que las de algunos productos dedicados como routers y filrewalls.

La herramienta IPROUTE2 es la encargada de permitirnos configurar y gestionar todo este conjunto de nuevas facilidades, además de llevar a cabo todo el trabajo que hasta ahora se hacía con las clásicas herramientas *ifconfig* y *route*. IPROUTE2 proporciona básicamente dos herramientas con las que implementar todas estas nuevas facilidades:

- La primera de ellas es la herramienta *ip*, de la que veremos algunas de sus posibilidades a continuación.
- La segunda es la herramienta *tc* (traffic control) que es la que permitirá implementar toda la Gestión de Tráfico y que trataremos en profundidad en posteriores apartados.

Como muestra de alguna de estas nuevas facilidades, mostraremos las múltiples tablas de enrutado. Sin embargo existen muchas más que no se detallarán por no estar dentro del ámbito de este trabajo, como:

- Túneles GRE y otros.
- Enrutado multicast.
- Trunking 802.1q
- ...

2.1. MÚLTIPLES TABLAS DE ENRUTADO.

En el nuevo diseño del subsistema de networking para Linux es posible definir varias tablas de enrutado, de forma que se puede decidir qué tabla es utilizada por un determinado tráfico IP. Esto es, podemos clasificar nuestro tráfico IP y aplicar un conjunto distinto de reglas de enrutado para cada tipo de tráfico.
Por defecto hay tres tablas, cada una con una prioridad distinta, y que se aplican a todos los tráficos. Para comprobar en cada momento el número de tablas definidas en nuestro equipo, y como se clasifican los paquetes para ir a cada una de ellas ejecutaremos el siguiente comando de iproute2:

```
# ip rule list

0: from all lookup local
32766: from all lookup main
32767: from all lookup default
```

Vemos las tres tablas que siempre existen por defecto *local, main y default*. Vemos además como todas las tablas se aplican a todos los paquetes (from all), sin embargo, la primera en ser aplicada es la que tiene un número de prioridad menor (local).
Veamos ahora con un ejemplo una posible situación en la que se añadiría una nueva tabla de enrutado, y cómo definimos qué tipo de tráfico utilizaría esa tabla.

CONTROL DE TRÁFICO UTILIZANDO LINUX

Supongamos que tenemos un router Linux con dos enlaces a Internet, uno de ellos con mayor ancho de banda que el otro. Supongamos además que damos acceso a Internet a una red de clientes, de forma que proporcionamos dos tipos de servicio. El primero de ellos es un acceso sólo para consultar hotmail y el segundo es un servicio de acceso normal. Lógicamente el primer servicio es más barato que el segundo, pero ello supone que utilizará el acceso de menor ancho de banda.

La dirección del acceso con mayor ancho de banda es 212.64.94.251 y es un enlace PPP a la dirección 212.64.94.1. El acceso de menor ancho de banda tiene dirección 212.64.78.148 y es un enlace a la dirección 195.96.98.253. Además, nuestros clientes con acceso a hotmail tendrán una dirección de la red 192.168.10.0/25, mientras que los clientes con acceso normal tendrán una dirección de la red 192.168.10.128/25. Veamos pues cómo implementaríamos todo esto:

Creamos la tabla y generaremos una nueva regla a la que llamaremos ‘hotmail’:

```
# echo 200 hotmail >> /etc/iproute2/rt_tables
# ip rule add from 192.168.10.0/25 table hotmail
```

Si ahora listamos nuestras tablas:

```
# ip rule ls
0: from all lookup local
32765: from 192.168.10.0/25 lookup hotmail
32766: from all lookup main
32767: from all lookup default
```

Lo único que nos queda es añadir la ruta a la tabla ‘hotmail’ y reiniciar la caché de rutas:

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

Lo usuarios pertenecientes a la red 192.168.10.128/25 accederán a través del otro acceso a Internet, el cuál estará configurado como gateway en las tablas de defecto.

Con este simple ejemplo hemos intentado mostrar el gran potencial que pone Linux a nuestro alcance.

3. DISCIPLINAS DE COLAS PARA LA GESTIÓN DE TRÁFICO

3.1. CONCEPTO DE COLA Y DISCIPLINA DE COLAS

Como hemos dicho en apartados anteriores, con las versiones del kernel de Linux 2.2.X y 2.4.X, las posibilidades dentro del ámbito de networking son enormes. Concretamente, dentro del campo de la Gestión de Ancho de Banda, Linux ofrece posibilidades que no tienen nada que envidiar a las de sistemas dedicados que podemos encontrar en el mercado.

Como sabemos, Internet utiliza la pila de protocolos TCP/IP. Sin embargo, TCP/IP no tiene forma de conocer la capacidad (en ancho de banda) de la red que une dos equipos que se comunican. De esta forma lo que ocurre es que los paquetes son enviados cada vez más rápido hasta que empieza a perderse parte de los mismos debido a que se supera la capacidad de la red, momento que TCP/IP aprovecha para ajustar la velocidad de envío.
Esta forma de trabajar es lo que se puede controlar y modificar mediante el proceso de encolar paquetes. Las colas y disciplinas de colas son, junto con otras, las herramientas que nos permiten definir las políticas de los procesos de encolar paquetes.
Antes de comenzar a definir tipos de disciplinas de colas, clases, filtros y demás, es aconsejable que fijemos algunos de esos conceptos:

.- **Disciplina de colas**
Es el algoritmo que gestiona el proceso de encolar paquetes en un dispositivo (interfaz de red). Esta gestión puede ser tanto en la cola de entrada (ingress), como en la cola de salida (egress).

.- Disciplina de colas sin clases
Es aquella disciplina de colas que no admite una subdivisión interna que pueda ser configurada por el usuario.

.- Clases y Disciplinas de colas con clases

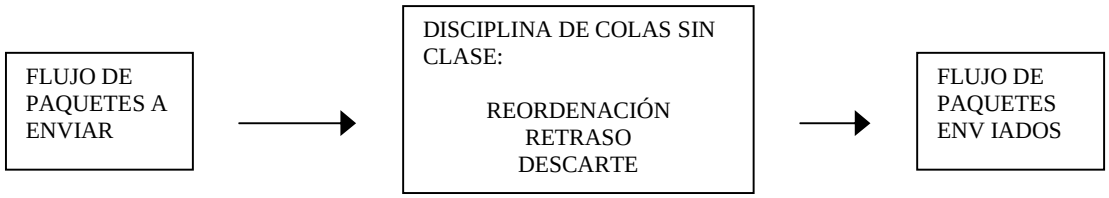
Una disciplina de colas con clases puede contener muchas clases, cada una de las cuales es interna a la disciplina de colas. Además, cada clase contiene una nueva disciplina de colas, que puede ser con clases o sin ellas.

.- Clasificador y filtro

Las disciplinas de colas con clases necesitan determinar a qué clase envían cada paquete que llega. Esto lo hacen utilizando un clasificador. A su vez la clasificación la llevan a cabo utilizando filtros, los cuales determinan una serie de condiciones que deben cumplir los paquetes.

3.2. DISCIPLINAS DE COLAS SIN CLASES.

Estos son los procesos de encolar paquetes más sencillos, pues sólo tienen la capacidad de reordenar, retrasar o descartar los paquetes que van llegando para ser enviados:



A continuación veremos los tres procesos de encolar paquetes que existen de este tipo:

.- PFIFO_FAST

.- TOKEN BUCKET FILTER

.- STOCHASTIC FAIRNESS QUEUEING

3.2.1. pfifo_fast

Esta disciplina de colas está formada por tres bandas (bandas 0, 1 y 2). Dentro de cada banda los paquetes son enviados siguiendo una política FIFO (First In, First Out). Sin embargo, ningún paquete de la banda 1 es enviado mientras existan paquetes por enviar en la banda 0, y lo mismo ocurre para las bandas 2 y 1. Es decir, existe una prioridad definida entre dichas bandas, siendo la banda 0 la más prioritaria y la banda 2 la de menor prioridad.
Para determinar los paquetes que van a cada banda se utiliza el campo TOS (Type Of Service) de la cabecera IP del mismo, tal y cómo veremos a continuación.
En la definición de disciplina de colas sin clases se especificó que éstas no tenían ninguna subdivisión interna en su estructura. Sin embargo, vemos como esta disciplina pfifo_fast sí que la tiene (posee tres bandas). Lo importante aquí es que una disciplina de colas sin clases no puede tener ninguna subdivisión interna de su estructura, susceptible de ser configurada por el usuario. Así pues, aunque la pfifo_fast tiene subdivisión interna, esta no puede ser modificada por el usuario.
Como hemos dicho, la forma en que los paquetes son distribuidos entre las tres bandas se basa en el campo TOS de la cabecera IP. El proceso consiste en que el kernel de Linux primero da una determinada prioridad a los paquetes en función del valor del campo, y posteriormente existe una mapeo de prioridades, **definibles por el usuario**, entre la prioridad asignada por el kernel y cada una de las tres bandas.

Los valores más importantes para el campo TOS, junto con su significado, son los siguientes:

Binario	Decimal	Significado
1000	8	Minimize delay (md)
0100	4	Maximize throughput (mt)
0010	2	Maximize reliability (mr)
0001	1	Minimize monetary cost (mmc)
0000	0	Normal Service

El mapeo de prioridades configurados por defecto es el siguiente:

Valor TOS	Prioridad Linux	Banda pfifo_fast
0x0 0	0 Best Effort	1
0x2 1	1 Filler	2
0x4 2	0 Best Effort	1
0x6 3	0 Best Effort	1
0x8 4	2 Bulk	2
0xa 5	2 Bulk	2
0xc 6	2 Bulk	2
0xe 7	2 Bulk	2
0x10 8	6 Interactive	0
0x12 9	6 Interactive	0
0x14 10	6 Interactive	0
0x16 11	6 Interactive	0
0x18 12	4 Int. Bulk	1
0x1a 13	4 Int. Bulk	1
0x1c 14	4 Int. Bulk	1
0x1e 15	4 Int. Bulk	1

Vemos como la última columna es la que determina cómo se mapea cada una de las prioridades del kernel de Linux a las bandas de la disciplina de colas.
Tal y como hemos dicho, este mapeo es configurable por el usuario a través del parámetro **priomap.**

3.2.2. Token Bucket Filter

Este tipo de disciplina de colas es la que debemos escoger en el caso de que nuestra única necesidad sea limitar el ancho de banda de un determinado interfaz.

El modelo de funcionamiento consiste en suponer que tenemos un buffer (bucket) al cual llegan los denominados ‘tokens’ a un ritmo constante. Estos tokens además serán los que utilizarán los paquetes IP para salir del interfaz de red. Es como si cada paquete IP tuviese que esperar a una carretilla (token) que será la encargada de sacarlo del interfaz. Así pues,

en función de cómo sean los ritmos de llegada de tokens y paquetes IP las situaciones que pueden plantearse son tres:

1.- Los paquetes IP llegan a mismo ritmo que los tokens. En este caso, cada paquete IP es asignado automáticamente a una de las carretillas que lo sacará del interfaz.

2.- Los paquetes IP llegan a un ritmo mayor que el de los tokens. En este caso, los paquetes IP tendrán que esperar durante un tiempo a que haya disponible una carretilla que les pueda sacar. Si esta situación se prolonga en el tiempo, parte de los paquetes IP que esperan empezarán a ser descartados con lo cual limitamos el ancho de banda.

3.- Los paquetes IP llegan a un ritmo menor que el de los tokens. En este caso, cada paquete IP será asignado automáticamente a una carretilla que lo sacará del interfaz. Además los tokens, que no han sido utilizados para sacar ningún paquete IP, serán almacenados en el buffer (bucket) hasta alcanzar el límite del mismo. De esta forma, si cambiara la tendencia y empezaran a llegar paquetes IP a un mayor ritmo, se podrían utilizar estos tokens almacenados para sacar, de manera instantánea, parte de esos paquetes IP que llegan.

Este modelo es un poco simple aunque explica perfectamente la dinámica de este algoritmo de disciplina de colas. En realidad, los tokens tienen correspondencia con bytes y no con paquetes IP, pero el resto del modelo es bastante aproximado.

Un ejemplo de utilización sería el siguiente.
Supongamos que tenemos un interfaz de red (eth0) del que queremos limitar el ancho de banda de salida para que no sature a otro equipo que utilizamos como gateway de salida a Internet. Supongamos que queremos limitar el ancho de banda a 220 Kbits/seg. Sería algo tan fácil como:

```
# tc qdisc add dev eth0 root tbf rate 220kbit latency 50ms burst 1540
```

Con este comando indicamos que vamos a añadir una disciplina de colas en el interfaz eth0 (tc qdisc add dev eth0).Además le indicamos que esta disciplina es del tipo Token Bucket Filter (tbf), y con referencia a esto último también indicamos lo siguiente:

.- El ritmo al que llegarán los tokens a la disciplina de colas será de 220 Kbits/seg (rate 220kbit), y que por tanto, esté será el máximo ancho de banda de salida para el interfaz eth0.

.- El tiempo máximo que permanecerá un paquete IP esperando por un token será de 50 milisegundos (latency 50ms).

.- El tamaño del buffer (bucket) donde se almacenarán los tokens no utilizados será de 1540 bytes (burst 1540). Normalmente mientras mayor es el límite impuesto al ancho de banda, mayor deberá ser el tamaño de este bucket.

3.2.3. Stochastic Fairness Queueing

Este tipo de disciplina de colas intenta distribuir el ancho de banda de un determinado interfaz de red de la forma más justa posible.

Para ello esta disciplina implementa una política de Round Robin entre todos y cada uno de los flujos de comunicación establecidos en el interfaz, dando a cada uno la oportunidad de enviar sus paquetes por turnos. Un flujo de comunicación será cualquier sesión TCP o flujo UDP, y de esta forma lo que conseguimos es que ninguna comunicación impida al resto poder enviar parte de su información. Lógicamente esta disciplina de colas sólo tendrá sentido en aquellos interfaces que normalmente estén saturados y en los que no queramos que una determinada comunicación eclipse al resto.

3.3. DISCIPLINAS DE COLAS CON CLASES.

Como hemos dicho en apartados anteriores, este tipo de disciplinas de colas se caracterizan por tener una subdivisión interna de su estructura, susceptible de ser configurada por el usuario, lo cual las hace muy útiles cuando tenemos diferentes tipos de tráfico que necesitan diferentes tratamientos.

Cuando los paquetes IP llegan a una disciplina de este tipo, necesitan ser enviados a una de las clases que la componen, es decir, necesitan ser clasificados. Para realizar esta clasificación se consultan los filtros asociados a la disciplina de colas, los cuales devuelven un resultado que permite a la disciplina de colas determinar a qué clase debe ser enviado el paquete. Además, cada clase sabemos que tiene asociada una nueva disciplina de colas (con o sin clases), con lo que nuevas consultas a filtros pueden ser realizadas hasta conseguir clasificar el paquete completamente.

En Linux, cada interfaz de red tiene una disciplina de colas de salida (egress) llamada ‘root’, que es la primera de su estructura interna. Por defecto, si no se especifica otra cosa, esta disciplina es del tipo pfifo_fast. Además, a cada disciplina de colas le es asignado un ‘manejador’ que se utilizará en los comandos de configuración de dicha disciplina. Estos manejadores constan de dos partes, un ‘número mayor’ y un ‘número menor’ separados por ‘:’, así el manejador de la disciplina de colas ‘root’ es ‘1:0’. Normalmente el número menor del manejador de una disciplina de colas es siempre cero, y el número mayor de las clases adjuntas a una disciplina de colas debe coincidir con el número mayor de la misma.

3.3.1. Disciplina de colas PRIO

Esta disciplina de colas recuerda a la disciplina sin clases pfifo_fast, aunque es mucho más versátil y ofrece mayores posibilidades.

Por defecto esta disciplina de colas define tres clases, y cada una de ellas tiene asociada una nueva disciplina de colas con política FIFO. Entre las tres clases existe una prioridad de forma que mientras haya paquetes en la clase 1 no se envían paquetes de la clase 2, y lo mismo entre las clases 2 y 3. Vemos pues como hasta aquí es casi una copia de la clase pfifo_fast. Sin embargo, la gran diferencia radica en dos factores:

1.- En esta clase podemos definir los filtros que creamos necesarios, de forma que no estamos limitados a hacer una clasificación de los paquetes en función del campo TOS, sino que podemos hacer una clasificación todo lo compleja que queramos.
2.- Aunque por defecto cada una de las tres clases asociadas a la disciplina PRIO tienen una disciplina con política FIFO, en realidad podemos definir la disciplina de colas que queramos. Por tanto, podría ser por ejemplo una nueva disciplina con clases, filtros asociados, etc.

Parámetros de las disciplinas de colas PRIO

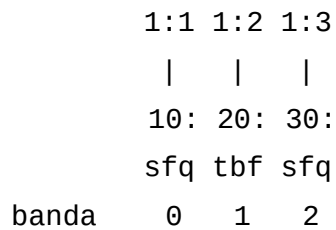
En este tipo de disciplinas se pueden configurar dos parámetros:

.- *bands.* Este parámetro permite especificar el número de clases (por defecto 3) que queremos que tenga la disciplina.

.- *priomap* En una disciplina de este tipo podemos decidirnos por no adjuntar ningún filtro que realice la clasificación del tráfico entre las distintas clases que tengamos. En ese caso la clasificación se hace siguiendo el mismo criterio (campo TOS) que en el caso de la disciplina pfifo_fast. Así, este parámetro permite determinar el mapeo que queremos establecer entre las prioridades del kernel de Linux, y nuestras clases.

Un ejemplo de utilización sería el siguiente.
Supongamos un interfaz en el que queremos dar prioridad al tráfico que necesita interactividad, frente al resto de tráfico.
Para ello crearíamos la siguiente estructura en la disciplina de colas:

```
root 1: prio
 /   |   \
```



El tráfico normal es enviado a la clase 30:, el tráfico que necesita interactividad es enviado a las clases 20: o 10:.

Los comandos a ejecutar serían los siguientes:

```
## Esta orden crea de forma intantánea las clases 1:1, 1:2, 1:3
##
# tc qdisc add dev eth0 root handle 1: prio
##
## A continuación adjuntamos las disciplinas de colas a cada una
de las tres clases

# tc qdisc add dev eth0 parent 1:1 handle 10: sfq
# tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate 20kbit buffer 1600
limit 3000
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq

## Ahora podríamos comprobar lo que hemos creado:

# tc -s qdisc ls dev eth0

qdisc sfq 30: quantum 1514b
Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
qdisc sfq 10: quantum 1514b
Sent 132 bytes 2 pkts (dropped 0, overlimits 0)
qdisc prio 1: bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
Sent 174 bytes 3 pkts (dropped 0, overlimits 0)
```

Si comprobamos este mapeo de prioridades entre el kernel de Linux y nuestras tres clases, veremos como efectivamente hemos conseguido lo que íbamos persiguiendo. Para ello es necesario recordar que la banda 0 corresponde a la clase 1:1, la banda 1 a la clases 1:2 y la banda 2 a la clase 1:3.

3.3.2. Disciplina de colas Class-Based Queueing (CBQ)

Esta disciplina de colas fue la primera que se creó, y probablemente la más utilizada. De hecho, en muchos ámbitos aún se asocia el Control de Tráfico en Linux exclusivamente en referencia a este tipo de disciplina de colas.

CBQ es la más compleja, menos entendida y más empírica de todas las disciplinas de colas. Esto obedece básicamente al hecho de que, además de ser una disciplina de colas con clase, CBQ ofrece la capacidad de regular el ancho de banda (shaping), siendo en este terreno donde surgen sus mayores dificultades.

Supongamos que se intenta reducir el ancho de banda de una conexión de red de 10 Mbit/seg a 1Mbit/seg. Para lograrlo tendremos que conseguir que el enlace esté inactivo el 90% del tiempo. Sin embargo, medir con exactitud ese porcentaje entraña mucha dificultad.

CBQ utiliza un algoritmo basado en la estimación del intervalo de tiempo transcurrido entre dos peticiones consecutivas del hardware para el envío de datos. Así, se ha comprobado que no siempre se consiguen buenas aproximaciones de este modo, e incluso a veces los resultados son totalmente equívocos.

No obstante, para la mayoría de las situaciones, este algoritmo trabaja de forma adecuada cumpliendo perfectamente con nuestras necesidades.

A continuación realizaremos un breve repaso sobre los parámetros más importantes de que se dispone a la hora de configurar este tipo de disciplinas de colas.

Parámetros CBQ para ajustar la regulación del ancho de banda.

Tal y como hemos dicho, CBQ trabaja intentando que el enlace esté inactivo durante un porcentaje de tiempo que asegure que se regula hasta conseguir el ancho de banda deseado.

También hemos hecho referencia a la complejidad del algoritmo y su falta de exactitud en determinadas situaciones. De hecho, el número de parámetros que se utilizan para esta tarea es elevado y a veces no se sabe cuál es exactamente su función, de ahí que en la mayoría de

ocasiones debe ser la propia experiencia la que enseñe como jugar con estos parámetros. Algunos de los más importantes son:

.- *avpkt*. Tamaño medio del paquete medido en bytes.

.- *bandwidth*. Ancho de banda del dispositivo físico. Se necesita para calcular el tiempo muerto entre petición y petición.

.- *mpu*. Tamaño mínimo de un paquete. Es necesario porque incluso un paquete de cero bytes de datos da lugar a una trama Ethernet de un tamaño mínimo distinto de cero.

.- *rate*. Ancho de banda regulado con el que queremos que funcione nuestra disciplina de colas

Parámetros CBQ para definir el comportamiento como disciplina de colas con clases.

Al igual que la disciplina PRIO, CBQ permite definir prioridades dentro de las clases que componen su estructura interna. De esta forma, cada vez que el nivel hardware solicita un paquete, CBQ lanza un proceso Round Robin por prioridades. Para la configuración de este proceso CBQ pone a disposición del usuario los siguientes parámetros:

.- *prio*. Establece las distintas prioridades entre las clases que componen la estructura interna de la disciplina de colas.

.- *allot y weight*. Ambos parámetros permiten configurar el hecho de que aquellas clases con un mayor ancho de banda puedan enviar mayor cantidad de información cada vez que les toque el turno durante el proceso de Round Robin por prioridades.

Parámetros CBQ para definir la posibilidad de prestar y pedir prestado ancho de banda entre las distintas clases.

Manteniendo siempre la limitación global en el ancho de banda establecido para la disciplina de colas CBQ, existe la posibilidad de que entre las clases se presten ancho de banda en el caso que sea posible. Los parámetros de que se dispone para ellos son:

.- *isolated/sharing.* Una clase configurada con el parámetro 'isolated' no prestará nunca ancho de banda a sus hermanas. El comportamiento contrario viene establecido por el parámetro 'sharing'. Por defecto, si no se indica lo contrario se supondrá que el 'sharing' está activo.

.- *bounded/borrow.* Una clase configurada con el parámetro 'bounded' no intentará pedir prestado ancho de banda a ninguna de sus hermanas. El comportamiento contrario viene establecido por el parámetro 'borrow'. Por defecto, si no se indica nada, se supondrá que el 'borrow' está activo.

Pronto veremos el apartado "Desarrollo de un caso real", en el que se expondrá un ejemplo de utilización de este tipo de disciplina de colas.

3.3.3. Disciplina de colas Hierarchical Token Bucket (HTB).

Como acabamos de ver, la disciplina de colas CBQ es compleja, y su comportamiento no es del todo óptimo en algunas situaciones. Básicamente CBQ es muy apropiada para casos en los que se dispone de un ancho de banda fijo que se quiera dividir en varios propósitos, dando a cada uno de ellos un ancho de banda garantizado, y con la posibilidad de prestar ancho de banda entre ellos.
HTB tiene una funcionalidad muy similar a la de CBQ, aunque su implementación es completamente distinta y su configuración mucho más sencilla. El único 'pero' es que HTB

aún no forma parte del kernel estándar de Linux y hay que parchearlo y recompilarlo para utilizarla.
La página del autor de esta disciplina de cola es: http://luxik.cdi.cz/~devik/qos/htb/.

3.4. UTILIZACIÓN DE FILTROS PARA LA CLASIFICACIÓN DE PAQUETES.

Sabemos que en las disciplinas de colas con clases es necesario llevar a cabo una clasificación del tráfico. Es decir, es necesario determinar a qué clase debe ir cada paquete IP.
Para ello utilizamos el concepto de 'filtro' que asociaremos a cada disciplina de colas en la que sea necesario llevar a cabo una clasificación.
Las posibilidades a la hora de filtrar los paquetes son múltiples, incluso hay algunos tipos de filtros que son específicos de determinadas disciplinas de colas. En este trabajo sólo haremos referencia a los dos tipos de filtros más significativos:

.- Filtro *u32*
.- Filtro *route*

3.4.1. Filtro u32.

Este tipo de filtro proporciona una versatilidad enorme al permitir muchos criterios a la hora de llevar a cabo el filtrado, lo cual hace que sean los más ampliamente utilizados.
Por definición, este tipo de filtro permite filtrar en función de cualquier conjunto de bits, tanto de la cabecera del paquete IP, como de la cabecera del segmento de datos. Sin embargo, este tipo de utilización es bastante farragosa y complicada, por lo que normalmente se suelen utilizar formas más directas para estos filtros. Así, algunas de estos criterios directos son:

.- Dirección IP de origen/destino del paquete.

.- Protocolo utilizado: tcp, udp, icmp, gre, ...

.- Puertos de origen y destino utilizados.

.- Valor del campo TOS de la cabecera IP.

Veamos algunos ejemplos:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 \
  match ip src 1.2.3.0/24 \
  match ip dst 4.3.2.0/24 flowid 10:1
```

Este filtro seleccionará aquellos paquetes IP que tengan como dirección IP origen cualquiera de la red 1.2.3.0/24 y como dirección destino cualquiera de la red 4.3.2.0/24.

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip src 4.3.2.1/32
  match ip dport 80 0xffff flowid 10:1
```

Este filtro seleccionará aquellos paquetes IP que tenga como dirección origen 4.3.2.1 y puerto destino el 80.

3.4.2. Filtro route.

Este tipo de filtros toman su decisión en función del resultado obtenido al pasar el paquete IP por la tabla de rutas.

Veamos un ejemplo:

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

Con este comando indicamos que los paquetes que vayan a cualquiera de las direcciones 192.168.10.0/24 será marcado con el ‘realm 10’ si son consultados por un filtro de tipo route.

Así, ahora podríamos utilizar el siguiente filtro:

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

Como vemos , cualquier paquete con destino 192.168.10.0/24 será enviado a la clase con manejador 1:10.

4. DESARROLLO DE UN CASO REAL.

A continuación expondremos el desarrollo de un caso real en el que se han utilizado algunas de las facilidades que Linux pone a nuestro alcance y que hemos ido analizando a lo largo de este trabajo.
Nuestra estructura de red se compone de dos sedes, una en Madrid y otra en Tarrasa. En cada una de las sedes hay una red plana tipo C y los aspectos más importantes de ambas son los siguientes:

1.- Tanto el servidor de correo electrónico como el acceso a Internet, para los equipos de ambas sedes, se encuentran en Tarrasa.
2.- En la sede de Madrid se encuentra el equipo servidor del Sistema de Gestión Documental, y en Tarrasa hay un equipo servidor de réplica. Ambos equipos mantienen una comunicación constante para sincronizar sus bases de datos.
3.- En la sede de Tarrasa se dispone de un servidor Unix que debe poder ser accedido desde la sede de Madrid.
4.- En la sede de Tarrasa existe un equipo que comunica esta sede con la red central de toda la corporación, y que también será accesible desde la sede de Madrid.
5.- Entre las sedes de Tarrasa y Madrid se ha contratado con un operador una línea punto a punto con un ancho de banda simétrico y asegurado de 512 Kilobits/seg.

El dibujo de toda esta estructura, incluyendo el direccionamiento IP, sería el siguiente:

CONTROL DE TRÁFICO UTILIZANDO LINUX

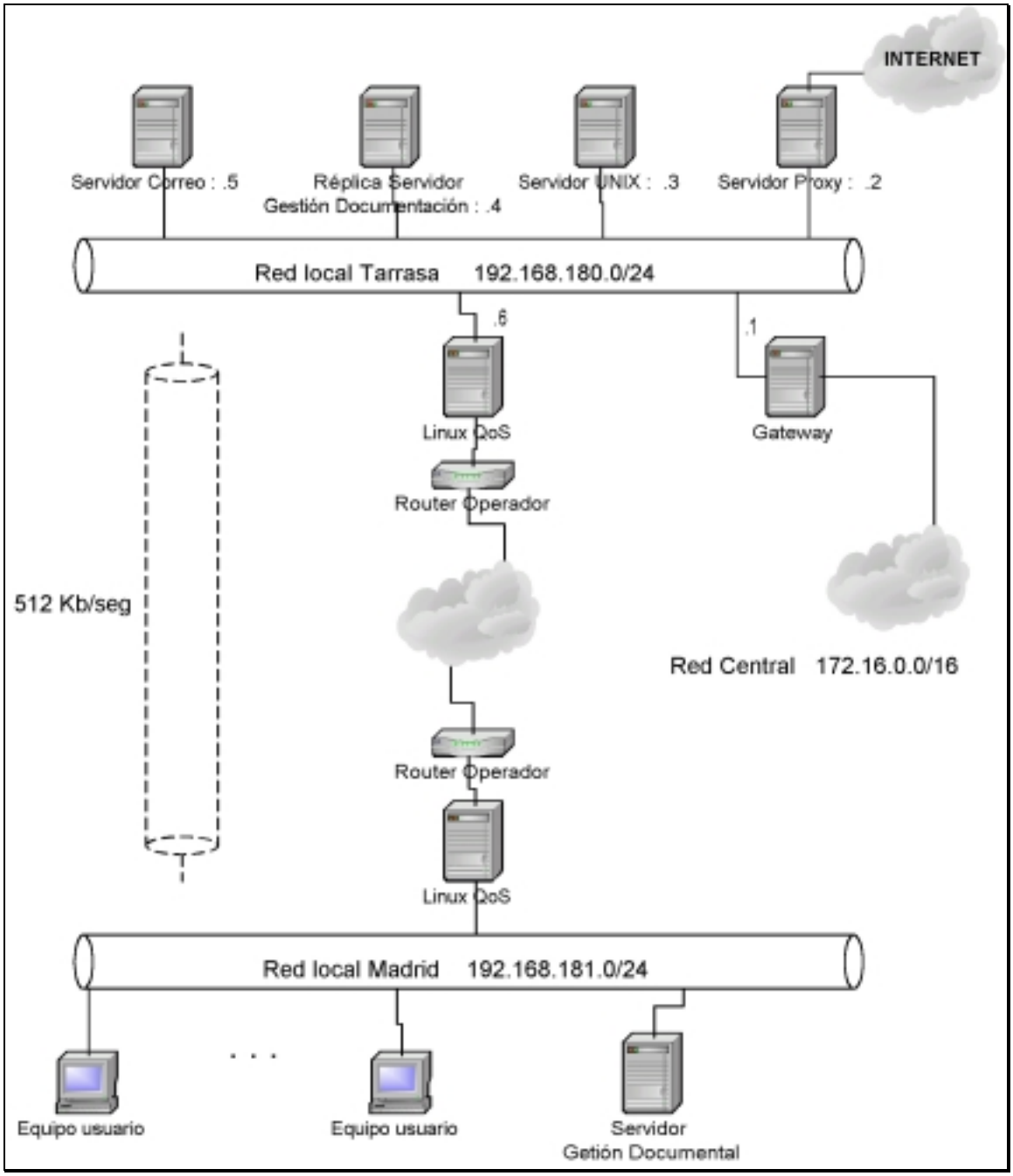


figura 1

CONTROL DE TRÁFICO UTILIZANDO LINUX

Tras identificar los distintos tráficos se decidió asignar el siguiente ancho de banda y prioridades entre ellos.

1.- Se decide asignar un ancho de banda de 64 Kbits/seg para la comunicación entre los Servidores de Gestión Documental. Es decir, cada vez que ambos servidores inicien una comunicación tendrán asegurado un mínimo de 64 Kbits/seg.
2.- Se decide asignar una ancho de banda mínimo de 128 Kbits/seg para la comunicación entre los usuarios de la sede de Madrid y el Servidor Unix de la sede de Tarrasa.
3.- El tráfico desde los usuario de la sede de Madrid hacia la red central de la corporación tendrá como mínimo un ancho de banda asegurado de 128 Kbits/seg.
4.- El resto del ancho de banda disponible se utilizará para el tráfico de correo electrónico e Internet.

Además, en caso de que alguno de los tráficos no esté utilizando todo el ancho de banda de que dispone, éste se repartirá entre el resto de los tráficos. Sin embargo, en este reparto tendrá prioridad el tráfico de correo e Internet. Así por ejemplo, si los Servidores de Gestión Documental no se estuvieran comunicando en un momento determinado, sus 64 Kbits/seg serían asignados automáticamente al tráfico de correo e Internet, y si éste no hiciera uso de ellos, entonces se repartirían a partes iguales entre el tráfico 2 y 3.

Para implementar esta distribución del ancho de banda se debe configurar de forma apropiada la estructura de colas de los equipos 'Linux QoS' de ambas redes. Concretamente la configuración se llevará a cabo en el interfaz de salida hacia el router del operador de ambos equipos (supondremos que ese interfaz es el eth0). Además, dicha configuración deberá ser simétrica en ambos equipos.
De esta forma el script para el equipo de la red de Madrid sería el siguiente:

```
#!/bin/sh
#
OPTIONS="bandwidth 100mbit allot 1514 maxburst 20 avpkt 1000 prio 3"
#
# Borramos cualquier posible configuración anterior
/sbin/tc del dev eth0 root
#
# Creamos la estructura de colas
/sbin/tc qdisc add dev eth0 root handle 10:0 cbq bandwidth 100mbit avpkt 100
/sbin/tc class add dev eth0 parent 10:0 classid 10:1 cbq bandwidth 100mbit \
rate 512kbit allot 1514 maxburst 20 avpkt 1000 bounded prio 3
/sbin/tc class add dev eth0 parent 10:1 classid 10:10 cbq rate 64kbit $OPTIONS
/sbin/tc class add dev eth0 parent 10:1 classid 10:20 cbq rate 128kbit $OPTIONS
/sbin/tc class add dev eth0 parent 10:1 classid 10:30 cbq rate 128kbit $OPTIONS
/sbin/tc class add dev eth0 parent 10:1 classid 10:40 cbq rate 192kbit \
bandwidth 100mbit allot 1514 maxburst 20 avpkt 1000 prio 2
#
# Establecemos la clasificación del tráfico
/sbin/tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 \
match ip dst 192.168.180.4/32 match ip src 192.168.181.4/32 flowid 10:10
/sbin/tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 \
match ip dst 192.168.180.3/32 flowid 10:20
/sbin/tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 \
match ip dst 172.16.0.0/16 flowid 10:30
/sbin/tc filter add dev eth0 parent 10:0 protocol ip prio 110 flowid 10:20
```

5. DOCUMENTACIÓN DE REFERENCIA.

.- **Marsh, Mattew G.** *Policy Routing Using Linux*

.- **Gregory Maxwell; Remco van Mook; Martijn van Oosterhout; Paul B Schroeder; Jasper Spaans.** *Linux Advanced Routing & Traffic Control HOWTO*

CONTROL DE TRÁFICO UTILIZANDO LINUX