Implantación de QoS

en un entorno GNU/Linux

Bruno Herrero (mines)

Última revisión: Octubre 2006

1. Introduccion

QoS (*Quality of Service o* Calidad de Servicio*)* es un conjunto de protocolos y tecnologías que **garantizan la entrega de datos a través de la red en un momento dado.** De este modo, nos aseguramos que las aplicaciones que requieran un tiempo de latencia bajo o un mayor consumo de ancho de banda, realmente dispongan de los recursos suficientes cuando los soliciten. Por ello, uno de las principales metas de QoS es la **priorización.** Esto es, el dar más relevancia a unas conexiones frente a otras.

Algunos de los beneficios que podemos obtener al implantar QoS en nuestro sistema son:

- **Control sobre los recursos:** podemos limitar el ancho de banda consumido por transferencias de FTP y dar más prioridad a un servidor de bases de datos al que acceden múltiples clientes.
- **Uso mas eficiente de los recursos de red:** al poder establecer prioridades dependiendo del tipo de servicio, umbrales de tasas de transferencia...
- **Menor latencia:** en aplicaciones de tráfico interactivo como SSH, telnet... que requieren un tiempo de respuesta corto.

Existen varias estrategias y técnicas para llevar a cabo la aplicación de QoS, tanto software como Hardware, comerciales y de código abierto (libres). En este documento se realizará la implantación práctica de un sistema QoS sencillo bajo un entorno Debian GNU/Linux. El propósito es proveer un acercamiento práctico al manejo de las disciplinas de cola bajo Linux que permiten ordenar el tráfico de red.

2. Caso de ejemplo

Para acercarnos un poco más a la filosofía de QoS, imaginemos un caso muy común. Tenemos un ordenador que está conectado a Internet a través de una línea ADSL con 1mb de bajada y 320kb de subida. Empezamos a generar tráfico de subida abundante, como por ejemplo mediante el uso de un programa P2P. ¿Por qué es imposible navegar por la web de forma fluida, si el tráfico de bajada y subida son independientes?. Por culpa de las colas.

Por defecto, cuando enviamos información por una interfaz (tarjeta de red en este caso) las tramas de datos **se encolan en una pila FIFO**: la primera trama en entrar, es la primera en salir. En nuestro caso, el módem que nos da salida a internet tiene una cola donde se colocan los paquetes para ser enviados, llamada Send-Q (*Send-Queue*, cola de envío). Al saturar esa cola con paquetes de envío P2P, será más difícil para los **paquetes ACK** de navegación web llegar a su destino, y esa es la causa de que no podamos navegar de forma fluida, porque el servidor HTTP no enviará mas datos hasta que no reciba las confirmaciones anteriores.

La solución para esto es trasladar la cola de envío a nuestro ordenador/router, de manera que podamos moldearla y darla forma a nuestro gusto, para gestionar el ancho de banda de manera más eficiente. En este caso, podremos decir que los paquetes dirigidos al puerto HTTP (80) tengan mucha más prioridad que los paquetes P2P; de esta forma saldrán antes por el módem los ACKs HTTP, permitiendo una mejor navegación al tener más prioridad que el tráfico P2P.

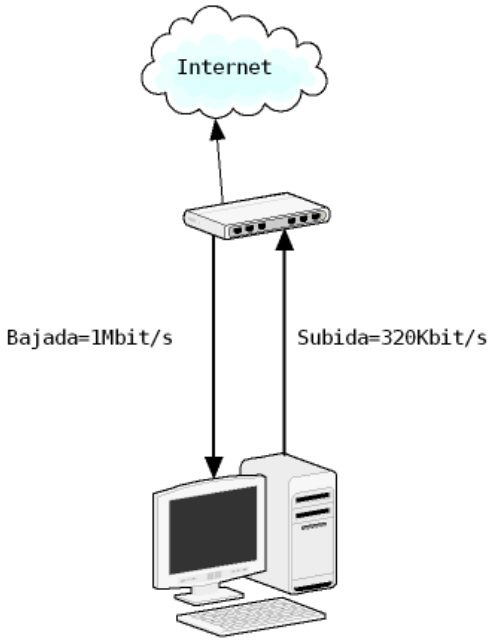

Fig. 1: Colas en el módem

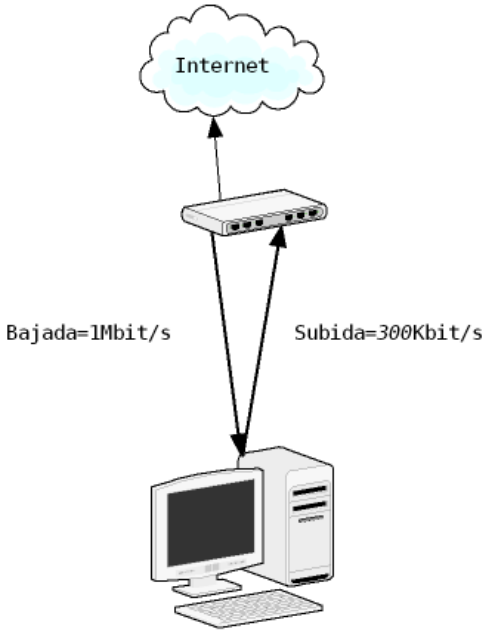

Fig. 2: Colas en el ordenador (router) con subida limitada

Debemos de entender la situación de la figura 2 como un “filtro” o semáforo regulador del tráfico: los paquetes se clasifican en el router (en nuestro caso el ordenador hará las veces de router) antes de llegar al módem. Además limitaremos **ligeramente por debajo** la velocidad de subida, de manera que **no saturemos** las colas de envío del modem. Con esto nos aseguramos de que no enviamos más datos de los que el módem puede manejar.

Este caso es extensible: en una red local corporativa con un gran número de puestos, puede ocurrir la situación de que parte de ellos comienzen a demandar ancho de banda en detrimento de servicios críticos que requieran alta disponibilidad y baja latencia, tales como tráfico interactivo SSH o servidores de aplicaciones. Aquí es donde entra en juega QoS y su capacidad para actuar como “agente” de la red, estableciendo prioridades de acceso, umbrales de transferencia y políticas para un control inteligente de los recursos de red.

3. Cómo funciona QoS en Linux

Para proceder a la implantación práctica de un sistema QoS para gestionar el ancho de banda, hay que comprender cual es el camino que recorre un paquete desde que "entra" o se genera en nuestra máquina Linux (que hará las veces de router) hasta que sale a Internet u otra red; así como las diferentes **disciplinas de cola** también conocidas como **qdiscs** (*Queue Disciplines*) que clasifican los paquetes.

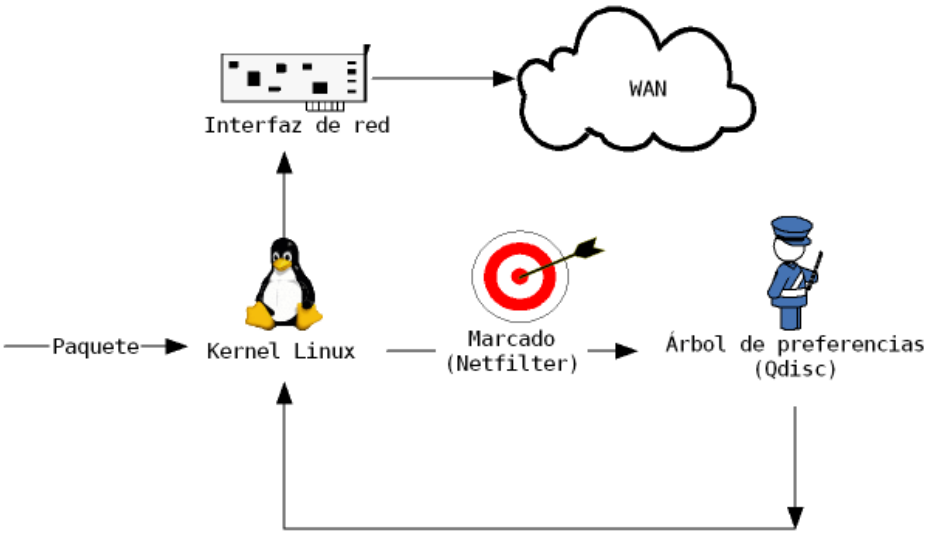

Fig. 3: Diagrama de paquetes con disciplinas de cola

En la figura 3 se puede observar de forma esquemática cual es el proceso que sigue un paquete de datos desde que llega hasta nuestra máquina local hasta que abandona la misma:

- El paquete llega al **kernel** linux de la máquina.
- **Netfilter** (iptables) se encarga de poner una **marca al paquete** que nosotros establezcamos. Es similar al código postal del sistema de correos: un identificador para posteriormente poder clasificar el envío. Se pueden hacer marcados en base al puerto de destino, a la cabecera IP, la dirección IP origen del paquete...
- El **árbol de preferencias** o árbol de preferencias: aquí es donde reside el corazón del mecanismo de control de tráfico. Con el encolamiento determinamos de que forma se envían los datos, mediante el uso de **disciplinas de cola.**

Disciplinas de cola

Las disciplinas de cola no son más que una estructura de clases en las que hay dependencia de padre-hijo entre sus miembros. Antes de entrar con detalle en ellas, es necesario conocer los algoritmos que hacen funcionar el mecanismo. Los aquí expuestos no son los únicos que existen, pero si los más comunmente usados.

- **ESFQ** (Enhaced Stochastic Fair Queing): el ancho de banda se reparte equitativamente entre todas las conexiones, dando la misma oportunidad a todas para enviar datos. Además, permite hacer el reparto en base a la dirección IP de la conexión. Esto es útil para controlar los programas P2P, que se aprovechan al crear múltiples conexiones.
- **HTB** (Hierarchical Token Bucket): permite dividir el ancho de banda disponible entre un mínimo y un máximo. El algoritmo asegura la disponibilidad del mínimo, y si se permite, alcanzar el máximo. Ademas puede "**pedir prestado"** el ancho de banda sobrante que no se use.
- **IMQ** (Intermediate Queing Device): se usa para encolar paquetes y limitar cuánto tráfico puede llegar a la interfaz de red. Tiene utilidad a la hora de limitar el tráfico **entrante.**
- **WRR** (Weighted Round Robin, Round Robin por Peso): similar al ESFQ, permite dar mas "peso" a una IP concreto y crear jerarquías balanceadas de tráfico.
- **PFIFO_FAST:** First In, First Out: el primero en entrar, es el primero en salir. Es el que está activado por defecto en linux, y el que se aplica también por defecto cuando creamos una clase.
- **RED** (Random Early Detection): utilizado para detectar la congestión. Se asegura de que la cola no se llene.
- **ECN** (Explicit Congestion Notification): funciona en conjunto con RED

Además, es preciso conocer algunos términos como:

- **Qdisc**: es el algoritmo que controla la cola de un dispositivo.
- **Qdisc raíz:** se encuentra adjunta a la interfaz de red.
- **Qdisc con clases:** permiten realizar subdivisiones internas, que a su vez pueden ser otras qdiscs.
- **Qdisc sin clases:** no se pueden configurar subdivisiones internas.
- **Clases:** una qdisc con clases puede tener múltiples clases, internas todas a la qdisc. A su vez también, pueden añadirse clases a clases. Las clases terminales son las que no tienen clases "hijas", y tienen una qdisc adjunta. Esta qdisc es la que se encarga de enviar los datos a la clase.

Cómo usar las disciplinas de cola para clasificar el tráfico.

Las disciplinas de cola siguen una estructura jerárquica en árbol. Cada interfaz (tarjeta de red) tiene una "qdisc raíz" que esta asociada a ella. Cuando se reciben paquetes o tramas de datos en la interfaz de red, la qdisc raíz recibe la petición de desencolar. En base a las marcas en el paquete que hayamos establecido con netfilter / iptables, la qdisc raíz lo enviará a alguna de las clases que contiene. A cada qdisc y cada clase se les asigna un controlador que consiste en dos partes separadas por dos puntos, un numero mayor y un número menor. Las clases deben tener el **mismo** número mayor que sus padres; el número menor debe ser único dentro de una qdisc y sus clases.

Para aclarar estos conceptos, se expone un ejemplo de implementación en la figura 4:

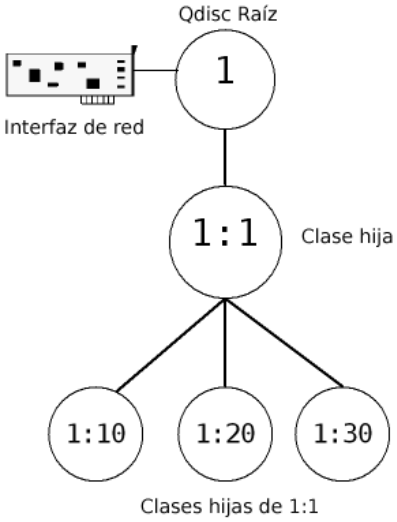

Fig. 4: Ejemplo de árbol

Las clases terminales 1:10, 1:20 y 1:30 son las que recibirán los paquetes. En cada una de ellas podemos establecer políticas diferentes: por ejemplo, la 10 puede tener más prioridad que la 20, la 30 tener más ancho de banda... Si no decimos nada, por defecto en cada clase terminal se asocia una qdisc FIFO.

Imaginemos de nuevo la conexión ADSL con 320kb de subida (se limitará a 300 por las razones anteriormente expuestas) que debe ser compartida entre dos ordenadores. Una posible solución para “compartir” el ancho de banda disponible sería la siguiente:

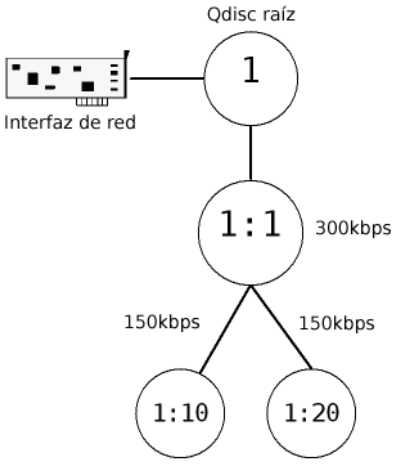

Fig. 5: Ejemplo de árbol con limitado de ancho de banda

Usando **HTB** podemos especificar que tanto la clase 1a clase 1:10 como la 1:20 dispongan de **al menos** 150kbit de ancho de banda de subida. Además, en el caso de que uno de los ordenadores no estuviese en funcionamiento, el otro podría alcanzar el tope de ancho de banda(lo “toma prestado”) que especifiquemos (en este caso, 300kbit). A cada una de las clases terminales también se las denomina **bandas.**

Si fuese necesario, podríamos también establecer **prioridades,** de manera que uno de los ordenadores tuviese más oportunidades de salir a la red que el otro.

4. Escenario del caso práctico

Antes de proceder a una implantación práctica, enumeremos cuáles son los elementos hardware y software disponibles en la práctica que se va a llevar a cabo, así como los **objetivos** que pretendemos con la implantación de QoS en nuestro router Linux.

- La red local es de clase A, con direcciones IP del tipo 10.0.0.0/255.0.0.0
- Un router-módem ADSL que da salida a Internet 1mbit/320kbit mediante Telefónica, con un hub de 4 bocas.
- Un servidor Debian Linux, **waham2**, que hara NAT en la red local para que salga a Internet y proveerá el servicio de QoS. Dispone de dos tarjetas de red. **Eth0** (IP 10,0,0,4) está conectada al router ADSL que da salida a internet, **eth1** (IP 10.0.0.3) a la red local, que a su vez está conectada a un hub donde se conectan los equipos. El kernel linux instalado es el 2,4,27-2.
- Un servidor Debian Linux, **waham** (IP 10,0,0,7) con Apache, servidor de correo (Cyrus y Postfix), servidor FTP proftpd y OpenSSH.
- Dos estaciones de trabajo: **waham5** (IP 10.0.0.5) con Windows 2000 SP 4, y **waham6** (IP 10,0,0,6) con Windows 98 SE.

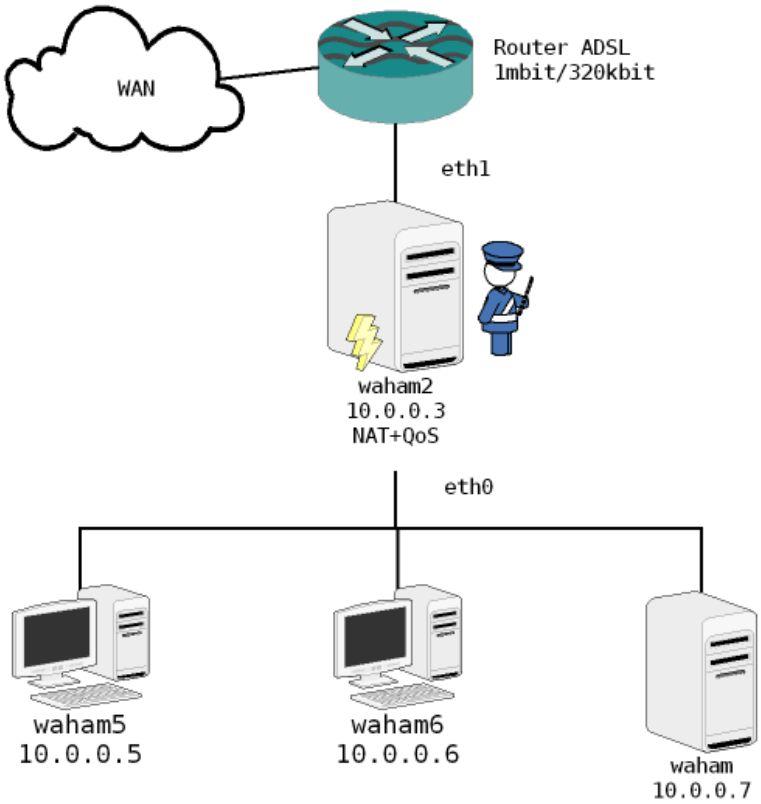

Fig. 6: Configuración de la red local

Objetivos

Si estuviésemos ante la necesidad de implantar QoS en una red con un gran número de equipos, se hace necesario un análisis exhaustivo y una planificación para determinar cuales son las necesidades de la red y de que manera podemos garantizar el mejor servicio. No obstante y en el caso que nos ocupa, no es necesario un análisis tan profundo, pero en todo caso es necesario conocer y saber de que manera queremos gestionar los recursos de red. La implantación QoS en nuestro caso práctico persigue:

- Dar más prioridad al servidor linux, ya que se trata de un servidor expuesto a Internet. Tendrá una salida a internet mínima de 200kbit. Los dos ordenadores restantes dispondrán de un caudal mínimo de 100kbit. En todo caso y si esta disponible, todos ellos tendrán como caudal el máximo permitido, 300kbit.
  Se realizará un reparto equitativo ESFQ del ancho de banda por IP y no por conexión.
- El tráfico SSH, ICMP, DNS, y los paquetes SYN (establecimiento de la conexión) y ACK tendrán más prioridad que el tráfico regular.
- El tráfico P2P y FTP llevará la prioridad más baja.
- El tráfico dirigido hacia el rango de puertos reservado (de 0 a 1024) irá en una cola de prioridad media (navegación web por ejemplo).

En todo caso hay que remarcar que **estamos perfilando el tráfico de subida y no el de bajada**, para limitar también el tráfico de bajada hay que hacer un pequeño “truco” que se verá más adelante.

5. Implantando QoS en Linux

Para instalar el sistema de control de tráfico en esta práctica, se ha necesitado:

- Última version del Kernel (actualmente 2,6,16,12). http://www.kernel.org
  Aunque la máquina tenía instalado el kernel 2,4,27, se hace necesario instalar el último disponible para dar soporte a las herramientas y parches que se usan aquí.
- Iproute2. Contiene la herramienta **tc** necesaria para configurar el árbol de preferencias (qdisc, clases...) http://linux-net.osdl.org/index.php/Iproute2
- Última versión de Iptables. http://www.netfilter.org
- Parche necesario para que el Kernel e Iproute puedan trabajar con ESFQ. http://fatooh.org/esfq-2.6/.
- Parche para poder usar IMQ (parche de kernel e iptables) http://www.linuximq.net/
- Layer7: muy útil para identificar protocolos P2P en base al contenido de los paquetes: http://l7-filter.sourceforge.net/. Parchea el kernel e iptables.

Parcheando y compilando el kernel

Una vez tengamos todos los parches y fuentes en nuestra máquina linux, es necesario parchear el kernel. Descomprimimos los archivos .tar.gz, por ejemplo, en nuestro directorio home (en este caso el de root), y nos situamos en el lugar donde hayamos descomprimido el kernel de linux. Normalmente instalaremos las fuentes del kernel en /usr/src/linux. Para parchear simplemente hay que introducir el comando `patch -p1 < /ruta_al_parche`:

```
waham2:/usr/src/linux# patch -p1 < ~/esfq-2.6.15.1/esfq-kernel.patch
waham2:/usr/src/linux# patch -p1 < ~/linux-2.6.16-imq1.diff
waham2:/usr/src/linux# patch -p1 < ~/netfilter-layer7-v2.2/kernel-2.6.13-2.6.16-layer7-2.2.patch
```

Cuando hayamos parcheado el kernel, ejecutamos `make menuconfig` para entrar en el menú n de configuración del kernel y configurarlo a nuestra voluntad, y además, **muy importante**, marcar las opciones necesarias o **no funcionará nada de lo que hagamos**:

- **Networking Options:** Network packet filtering
- **IP: Netfilter configuration:**
  - Netfilter MARK match support
  - Connection state match support
  - Packet filtering
  - Packet mangling
  - MARK target support
  - Connection tracking
  - Layer7 match support

Además, no hay que olvidarse de escoger también la opción **QoS and Fair Queing** dentro de Networking Options, como también aquellas cosas que vayamos a usar posteriormente, como por ejemplo **HTB Packet Scheduler,** si queremos usar htb (que si que usaremos en este caso).

Cuando hayamos terminado de configurar el kernel, sólo nos queda salir del menú de configuracion y compilarlo, junto a los módulos. Este documento no pretende ahondar en los pormenores de compilación del kernel. Si se desea más información al respecto, visitar http://www.insflug.org/COMOs/Kernel-Como/Kernel-Como.html

Parcheando y compilando iproute2 e iptables

Ya tenemos nuestro kernel listo, le hemos aplicado los parches correspondientes para que podamos usar ESFQ, IMQ y Layer7. Ahora sólo nos queda aplicar los parches al código fuente de iproute2 e iptables. El proceso es similar al anterior, usando el comando patch:

```
waham2:~/iptables-1.3.5# patch -p1 < ../iptables-1.3.0-imq1.diff
waham2:~/iptables-1.3.5#  patch  -p1  <  ../layer7/netfilter-layer7-v2.2/iptables-layer7-2.2.patch
```

Una vez parcheado nos queda hacer `make && make install` para compilar iptables e instalarlo. El proceso para iproute2 es similar.

Con el kernel, iproute2 e iptables convenientemente parcheados y compilados, sólo nos queda reiniciar la máquina para arrancar con el nuevo kernel (configuraremos grub para ello) y ya estamos listos para empezar a definir las políticas de tráfico.

El comando tc

El comando tc "Traffic Control", herramienta incluida dentro del paquete iproute2, será el programa para crear el árbol de bandas del que hemos hablado anteriormente. Su sintaxis es la siguiente:

Para gestionar qdiscs: `tc qdisc [ add | change | replace | link ] dev DEV [ parent qdisc-id | root ] [ handle qdisc-id ] qdisc [ qdisc specific parameters ]`

Para gestionar las clases: `tc class [ add | change | replace ] dev DEV parent qdisc-id [ classid class-id ] qdisc [ qdisc specific parameters ]`

Para gestionar los filtros: `tc filter [ add | change | replace ] dev DEV [ parent qdisc-id | root ] protocol protocol prio priority filtertype [ filtertype specific parameters ] flowid flow-id`

Muestra las qdiscs: `tc [-s | -d ] qdisc show [ dev DEV ]`

Muestra el tráfico de las clases: `tc [-s | -d ] class show dev DEV`

Muestra los filtros: `tc filter show dev DEV`

Primero crearemos la qdisc raiz. Es la principal y siempre tiene que estar presente; como hemos dicho es la que está asociada a la tarjeta de red. La tarjeta de red de salida a internet es eth1. Creamos la banda principal (root) con nombre 1:. Por defecto enviará los paquetes a la clase 13.

```
tc qdisc add dev eth0 root handle 1: htb default 13
```

Con root indicamos que queremos crear la qdisc raíz, que tendra el identificador 1:. Seguimos creando el árbol, en base a los objetivos expuestos en el punto 4.

```
tc class add dev eth0 parent 1: classid 1:1 htb rate 300kbit ceil 300kbit
```

Hemos definido una banda dependiente de root (parent 1:) con algoritmo htb. Esta clase no dejará pasar paquetes a más velocidad de 300kbit. Ceil hace referencia al **máximo posible** de transferencia, y rate al **mínimo garantizado.**

Ahora deberemos crear dos clases: una para el servidor linux (1:20), que garantize un ancho de banda mínimo de 200kbit, y otra que compartirán los dos ordenadores restantes (1:10), con un ancho de banda mínimo de 100kbit. La razón de que no hayamos creado estas clases directamente dependientes de la raíz 1:, es que **la qdisc raiz no puede prestar ancho de banda si sobra**, y por ello necesitamos crear la banda intermedia 1:1. Recordar que el servidor tiene más prioridad. Esta se puede definir mediante el indicador `prio`. Cuanto menor es el índice prio, mayor es la prioridad, y por defecto prio siempre es 0, con lo cual si no especificamos nada, crearemos clases con la prioridad más alta.

```
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 100kbit ceil 300kbit prio 1
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 200kbit ceil 300kbit
```

Repartiremos los paquetes en siete bandas, cada una de ellas contará con un mínimo de 14kbit (100kbit/7) y un máximo de 300kbit. (Cuanto más alto el numero, más baja es la prioridad)

- 0. Tráfico ICMP
- 1. Paquetes TCP con los flags SYN, ACK y RST y tráfico DNS
- 2. Tráfico por defecto
- 3. Rango de puertos reservados 0-1024
- 4. Tráfico HTTP.
- 5. Tráfico FTP
- 6. Tráfico P2P

Creamos las clases de la siguiente manera:

```
tc class add dev eth0 parent 1:10 classid 1:11 htb rate 14kbit ceil 300kbit
```

```
tc class add dev eth0 parent 1:10 classid 1:12 htb rate 14kbit ceil 300kbit prio 1
tc class add dev eth0 parent 1:10 classid 1:13 htb rate 14kbit ceil 300kbit prio 2
tc class add dev eth0 parent 1:10 classid 1:14 htb rate 14kbit ceil 300kbit prio 3
tc class add dev eth0 parent 1:10 classid 1:15 htb rate 14kbit ceil 300kbit prio 4
tc class add dev eth0 parent 1:10 classid 1:16 htb rate 14kbit ceil 300kbit prio 5
tc class add dev eth0 parent 1:10 classid 1:17 htb rate 14kbit ceil 300kbit prio 6
```

Necesitamos añadir qdisc ESFQ bajo las clases para que los dos ordenadores tengan las mismas oportunidades de salir a la red:

```
tc qdisc add dev eth0 parent 1:11 handle 11: esfq perturb 10 hash src
tc qdisc add dev eth0 parent 1:12 handle 12: esfq perturb 10 hash src
tc qdisc add dev eth0 parent 1:13 handle 13: esfq perturb 10 hash src
tc qdisc add dev eth0 parent 1:14 handle 14: esfq perturb 10 hash src
tc qdisc add dev eth0 parent 1:15 handle 15: esfq perturb 10 hash src
tc qdisc add dev eth0 parent 1:16 handle 16: esfq perturb 10 hash src
tc qdisc add dev eth0 parent 1:17 handle 17: esfq perturb 10 hash src
```

`Perturb 10` sirve para reconfigurar el *hash* cada 10 segundos, y `hash src` es el modificador para que el reparto se haga en base a la dirección IP de la conexión. Para mas información, acudir a la documentación de ESFQ.

De igual manera, creamos las clases para el servidor linux, con la salvedad de que en este caso no necesitamos una específica para encuadrar el tráfico P2P. El mínimo ahora es de 200kbit/7.

- 0. Tráfico ICMP
- 1. Paquetes TCP con los flags SYN, ACK y RST y tráfico DNS
- 2. SSH
- 3. Servidor Apache (puerto 80)
- 4. Servidor de correo SMTP (puerto 25) y servidor de correo Cyrus/IMAP (puerto 993)
- 5. Servidor FTP (puertos 20 y 21)
- 6. Resto de tráfico

```
tc class add dev eht0 parent 1:20 classid 1:21 htb rate 28kbit ceil 300kbit
tc class add dev eth0 parent 1:20 classid 1:22 htb rate 28kbit ceil 300kbit prio 1
tc class add dev eth0 parent 1:20 classid 1:23 htb rate 28kbit ceil 300kbit prio 2
tc class add dev eth0 parent 1:20 classid 1:24 htb rate 28kbit ceil 300kbit prio 3
tc class add dev eth0 parent 1:20 classid 1:25 htb rate 28kbit ceil 300kbit prio 4
tc class add dev eth0 parent 1:20 classid 1:26 htb rate 28kbit ceil 300kbit prio 5
tc class add dev eth0 parent 1:20 classid 1:27 htb rate 28kbit ceil 300kbit prio 6
```

Añadiremos una qdisc, esta vez SFQ (no nos hace falta ESFQ porque sólo disponemos de una IP en este “canal” o banda).:

```
tc qdisc add dev eth0 parent 1:21 handle 21: sfq perturb 10
tc qdisc add dev eth0 parent 1:22 handle 22: sfq perturb 10
tc qdisc add dev eth0 parent 1:23 handle 23: sfq perturb 10
tc qdisc add dev eth0 parent 1:24 handle 24: sfq perturb 10
tc qdisc add dev eth0 parent 1:25 handle 25: sfq perturb 10
```

Ya tenemos la estructura para clasificar el tráfico de salida de nuestra interfaz. Esta sería su apariencia:

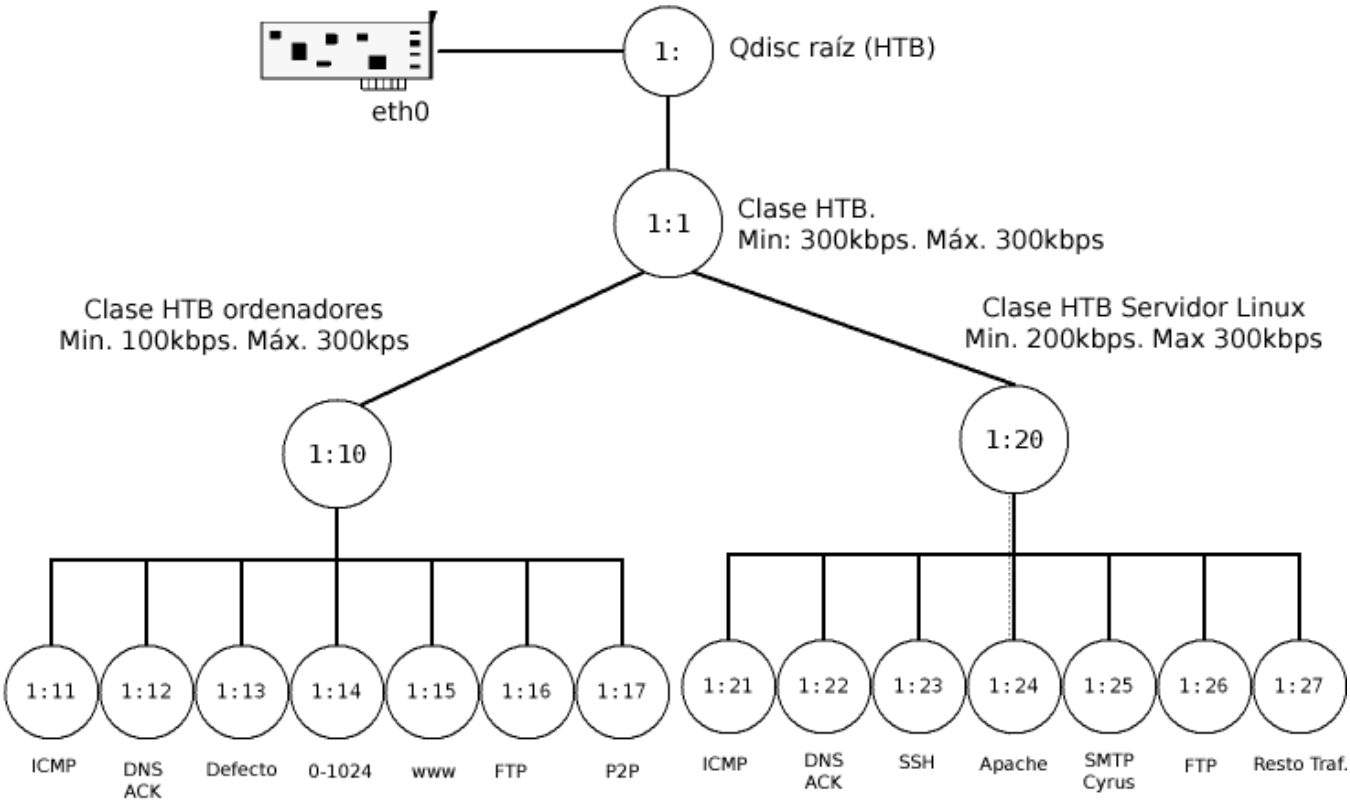

Fig. 7: Árbol de encolado

Marcado de paquetes con iptables.

El árbol que hemos creado, de por si no es capaz de priorizar el tráfico. Hemos de usar el firewall de linux (iptables) para hacer unas “marcas” a los paquetes con el fin de que posteriormente la qdisc raíz envíe el tráfico a las clases adecuadas.

Recordamos que estamos en el servidor que ofrece QoS opera una NAT, por eso habrá que marcar los paquetes en la regla FORWARD de iptables para los ordenadores.

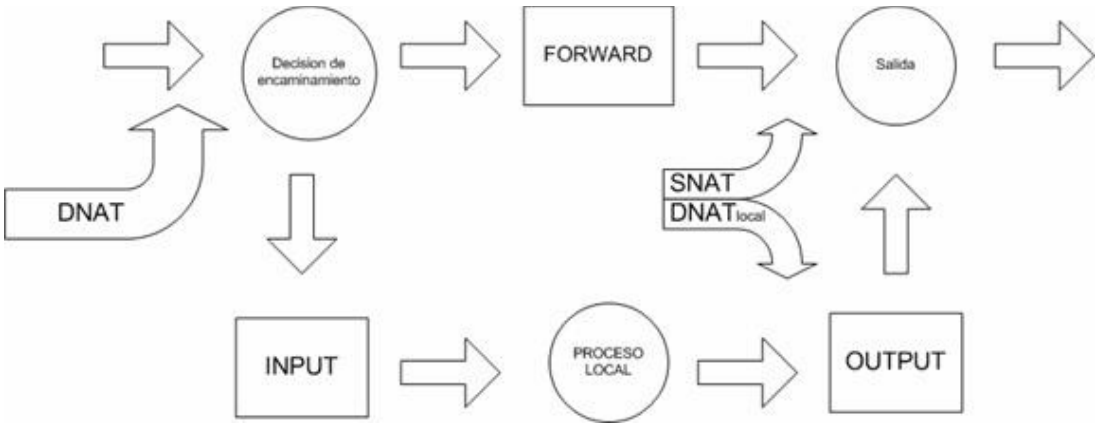

Fig. 8: Diagrama de paquetes del kernel

Es necesario conocer a fondo el porqué de marcar los paquetes en forward, output, input.. Se da por hecho que se tienen unos conocimientos básicos acerca del tema. Se puede obtener información sobre el manejo de iptables en http://www.pello.info/filez/firewall/iptables.html

Primero vamos a marcar los paquetes que vengan del servidor linux:

```
iptables -t mangle -A FORWARD -i eth1 -p icmp -s 10.0.0.2 -j MARK --set-mark 21
iptables -t mangle -A FORWARD -i eth1 -p icmp -s 10.0.0.2 -j RETURN
iptables -t mangle -A FORWARD -i eth1 -p tcp -m tcp --tcp-flags SYN,RST,ACK ACK -s
10.0.0.2 -j MARK --set-mark 22
iptables -t mangle -A FORWARD -i eth1 -p tcp -m tcp --tcp-flags SYN,RST,ACK ACK -s
10.0.0.2 -j RETURN
iptables -t mangle -A FORWARD -i eth1 -p tcp -m tcp --dport 53 -s 10.0.0.2 -j MARK
--set-mark 22
iptables -t mangle -A FORWARD -i eth1 -p tcp -m tcp --dport 53 -s 10.0.0.2 -j RETURN
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 22 -j MARK --
set-mark 23
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 22 -RETURN
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 80 -j MARK --
set-mark 24
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 80 -j RETURN
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 25 -j MARK --
set-mark 25
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 25 -j RETURN
--set-mark 25
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 993 -j MARK
--set-mark 25
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 993 -j RETURN
--set-mark 25
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 21 -j MARK --
set-mark 26
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 21 -RETURN
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 20 -j MARK --
set-mark 26
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 --source-port 20 -j RETURN
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 -j MARK --set-mark 27
iptables -t mangle -A FORWARD -i eth1 -p tcp -s 10.0.0.2 -j RETURN
```

Utilizamos la tabla mangle de iptables para el marcado, dentro de la regla de FORWARD. Con -j MARK y –set-mark [id] le ponemos el número de marca al paquete. Se usa RETURN para evitar la

coincidencia múltiple de paquetes. El marcado para los ordenadores sería similar. La única diferencia es que usaremos layer7 para el marcado de los paquetes P2P de la siguiente forma:

```
iptables -t mangle -A FORWARD -i eth1 -p tcp -m layer7 --l7proto edonkey -j MARK --
set-mark 17
```

Muy importante: no hay que olvidar habilitar **NAT** masquerading para los ordenadores, de lo contrario **no** saldrán a internet. De igual manera hay que habilitar el soporte para fowarding que permita que los paquetes pasen "a través" de nuestra máquina linux y configurar las rutas adecuadamente, para que los paquetes vayan a la tarjeta de red correcta:

```
iptables -t nat -A POSTROUTING -s 10.0.0.0/8 -j MASQUERADE
echo 1 > /proc/sys/net/ipv4/ip_forward
route add -net 10.0.0.0 netmask 255.0.0.0 dev eth1
route add default gw 10.0.0.2
```

Asociación de marcas con clases

Ya tenemos creado el árbol de preferencias, y iptables está configurado para marcar los paquetes. Sólo nos queda el "puente" de unión que no es otro que los **filtros** que tenemos que configurar dentro de la qdisc raíz para que envíe los paquetes a las clases:

```
tc filter add dev eth0 protocol ip parent 1: handle 21 fw classid 1:21
tc filter add dev eth0 protocol ip parent 1: handle 22 fw classid 1:22
tc filter add dev eth0 protocol ip parent 1: handle 23 fw classid 1:23
tc filter add dev eth0 protocol ip parent 1: handle 24 fw classid 1:24
tc filter add dev eth0 protocol ip parent 1: handle 25 fw classid 1:25
tc filter add dev eth0 protocol ip parent 1: handle 26 fw classid 1:26
tc filter add dev eth0 protocol ip parent 1: handle 27 fw classid 1:27
```

Se usa el modificador **fw** para que se clasifique en base a la marca del paquete hecha en iptables, se puede usar **u32** que nos permitiría hacer clasificaciones en base a la cabecera IP, pero es más complejo y tedioso. Habría que configurar igualmente los filtros para los dos ordenadores restantes, pero el proceso es idéntico; cambiando lógicamente los *handles*.

A modo de resumen aclaratorio, imaginemos que ocurriría si el ordenador 10.0.0.5 intentase conectarse al emule:

- El programa intenta acceder a la red. Para ello accede a Internet a través de la puerta de enlace (waham2) donde hemos instalado QoS.
- Los paquetes coinciden en la regla de iptables que hemos marcado con layer7, poniendoles la marca **17.**
- Los paquetes llegan a la **qdisc** raíz.
- Desde dentro de la qdisc, se llama a los filtros. Mira la marca del paquete (17) y lo envía a la clase correspondiente 1:17
- El paquete sale hacia internet por eth0, con las limitaciones que hemos impuesto en 1:17:

prioridad 6 (si hay por ejemplo paquetes de tráfico HTTP, saldrán antes estos).

Comprobando el funcionamiento

Para comprobar que lo que hemos hecho funciona, podemos verificar como si en el ordenador 10.0.0.5 usamos algún cliente eMule, y nos ponemos a navegar por web, el P2P no afectará a la navegación por tener menos prioridad, y podremos ver páginas web fluidamente.

Además, disponemos de las herramientas de tc para ver el tráfico en las clases. Con el comando `tc -s -d class show dev eth0` podemos verlo. Por ejemplo, esto es lo que mostraría cuando en el odenador 10.0.0.5 hemos navegado por alguna página web, nada más haber implantado el árbol de preferencias:

```
class htb 1:14 parent 1:10 leaf 14: prio 3 quantum 1400 rate 112000bit ceil 2400Kbit
burst 1655b/8 mpu 0b overhead 0b cburst 2799b/8 mpu 0b overhead 0b level 0
 Sent 21457 bytes 150 pkt (dropped 0, overlimits 0 requeues 0)
 rate 424bit 0pps backlog 0b 0p requeues 0
 lended: 150 borrowed: 0 giants: 0
 tokens: 102286 ctokens: 8587
```

Se han enviado 21,457 bytes hacia el exterior, con destino puerto 80. (Está clasificado para ir a 1:14). También podemos ver el tráfico en las qdiscs con `tc -s qdisc ls dev eth0`.

5. Regular el tráfico de entrada

Todos los pasos que hemos seguido han servido para dar forma al tráfico de subida, pero, ¿que ocurre con el tráfico de bajada?. La respuesta rápida es que no podemos, al menos de forma inmediata, limitar el tráfico entrante. Podemos controlar lo que enviamos a internet, pero no lo que nos llega de internet, del mismo modo que podemos controlar el correo electrónico que enviamos, pero no el que recibimos.

Hay una manera de limitar este tráfico, y es usando un pequeño “truco”: descartar paquetes. El protocolo TCP basa su funcionamiento en “arranque lento”: esto significa que va adquiriendo velocidad paulatinamente hasta que el host receptor no puede aceptar más paquetes y los empieza a descartar. Usando el dispositivo virtual de encolado **IMQ** podemos limitar este tráfico. Aún así, no se trata de un sistema perfecto.

Hay sistemas comerciales de QoS que se basan en la modificación del tamaño de la ventana para reducir el tráfico entrante.

Un ejemplo sencillo de lo que podríamos hacer con IMQ. Por ejemplo: limitar las descargas HTTP a 128kbit:

```
# Transladamos todo el tráfico entrante al dispositivo IMQ
iptables -t mangle -A POSTROUTING -o eth0 -j IMQ --todev 0
# Qdisc para IMQ con una clase con tope 128kbit
tc qdisc add dev imq0 handle 1: root htb default 1
tc class add dev imq0 parent 1: classid 1:1 htb rate 128kbit
tc class add dev imq0 parent 1:1 classid 1:11 htb rate 128kbit
# Regla iptables para marcado HTTP
iptables -t mangle -A PREROUTING -i imq0 -p tcp –sport 80 -j MARK --set-mark 11
# Asociacion
tc filter add dev imq0 parent 1:0 protocol ip handle 11 fw classid 1:11
```

6. Bibliografía

- Linux Advanced Routing & Traffic Control: http://www.lartc.org/
- Uso de QoS para equilibrar tráfico por IP y limitar tráfico P2P: http://jopi.seriousworks.net/item/qos_en_linux
- Introducción a Control del trafico y a Calidad de servicio en GNU/Linux: http://www.etxea.net/docu/qos/qos.html
- TCP/IP. Autor: Dr. Sidnie Feit. Editorial Osborne McGraw-Hill. ISBN 84-481-1531-7
- Quality of service networking:

  http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm#wp1024961
- Formación de colas y prioridades de paquetes: http://www.openbsd.org/faq/pf/es/queueing.html
- Modelo de evaluación de QoS para una red de campus: http://www.informatica.uagrm.edu.bo/materias/inf423/2005/ii/sb/mendozamm/Investigaci%C3%B3n/QoS.htm
- Método para la ecualización del ancho de banda: http://bulma.net/body.phtml?nIdNoticia=1727
- Entrada QoS en la Wikipedia inglesa: http://en.wikipedia.org/wiki/Quality_of_service
- Filtrado de paquetes y QoS: http://redes-linux.all-inone.net/manuales/ancho_banda/filter_qos.pdf
- Routing avanzado con el núcleo Linux. Reserva de ancho de banda: http://congreso.hispalinux.es/congreso2001/actividades/ponencias/eric/html/banda.html