QUALITY OF SERVICE

Carlos Álvarez Martín
Proyecto Integrado ASI 2009/2010
I.E.S. Gonzalo Nazareno

Índice de contenido

1. Introducción

Dentro del ámbito de la informática, podemos definir QoS (Quality of Service) como un conjunto de mecanismos con los que se trata de conseguir dar buen servicio de forma ininterrumpida. El QoS se puede aplicar en redes WiFi, conexiones ATM, etc. En este caso se implantará sobre una red de más de 50 puestos de trabajo con una conexión ADSL.

Consiste en priorizar cierto tráfico así como ajustar el ancho de banda para cada uno de los protocolos que tratemos en concreto, el de la línea de forma global, etc.
Intentaremos, aplicando una serie de técnicas, que nuestra conexión funcione lo más fluida posible en situaciones de mucha demanda.

Existen implementaciones QoS tanto software como hardware (la mayoría de las veces se tratan de dispositivos corriendo un kérnel Linux).
En nuestro caso será una implementación software.

Digamos que QoS se podría representar con la siguiente igualdad:

QoS = Priorización + Moldeado

2. Funcionamiento de QoS en Debian (GNU/Linux)

A la hora de hacer QoS se pueden usar distintas soluciones o formas de abordar el problema. Un ejemplo sería el proxy Squid para realizar QoS a partir de ACLs y las llamadas “Delay Pools”. Sin embargo en nuestro caso, debido a su popularidad y a un uso de recursos más liviano, emplearemos un conjunto de dos herramientas. Se tratan de *TC* (incluida en el paquete Iproute2) e *IPTABLES*.

Para poder usar dichas herramientas en nuestro servidor, hemos de comprobar que nuestro kérnel está compilado con soporte para QoS y filtrado de paquetes (Iptables). Estas características se encuentran en “Networking/Networking options” dentro del menú de configuración del kérnel (secciones “Network packet filtering framework” y “QoS and/or fair queueing”). En nuestro caso, usando Debian Lenny (kérnel 2.6.26) no tendremos que recompilar el núcleo puesto que todas las características ya mencionadas se encuentran incluidas en él.

La forma en la que trabajará nuestro sistema de QoS (TC e Iptables) será la siguiente:

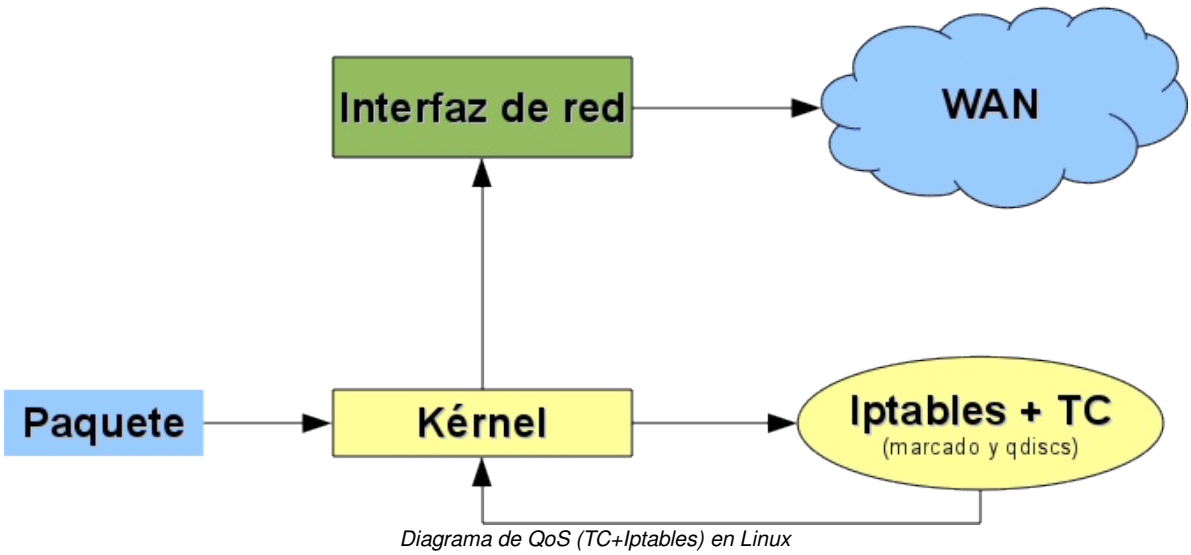


Diagrama de QoS (TC+Iptables) en Linux

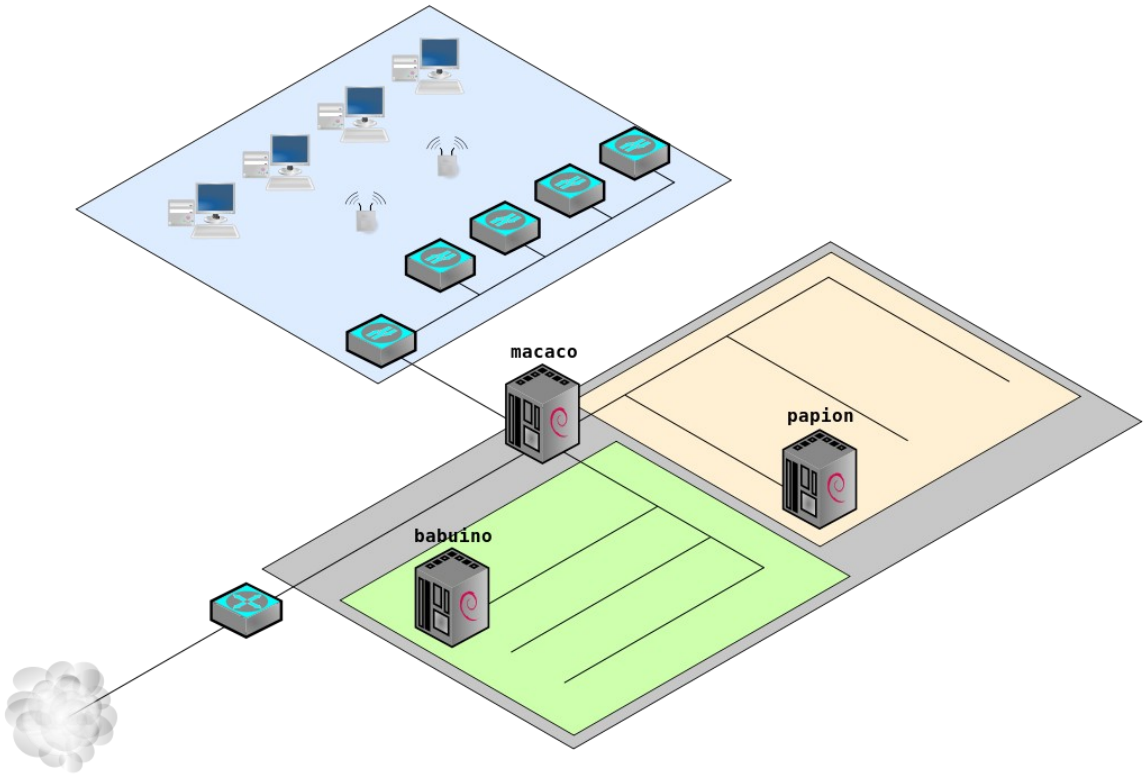
Lo que pretendemos es que en lugar de que le módem (o router doméstico) tenga la cola

de nuestra línea, sea nuestro núcleo Linux el que se encargue de dicha función (“hay que poseer la cola”). Es relativamente fácil hacerlo con las herramientas que usaremos, y se consigue ajustando las velocidades de subida y descarga a unos límites un poco inferiores a los alcanzables por nuestra conexión real (nuestro enlace con el ISP). Además, conseguiremos “liberar” un poco al módem que usamos para conectar con el ISP ya que suelen congestionarse con situaciones de tráfico elevado.

3. Caso práctico

En esta sección se muestra cómo se ha implantado el QoS en los ciclos de informática del I.E.S. Gonzalo Nazareno. Se tomó dicha decisión debido a la pobre calidad de conexión de la que disponían tanto alumnos como profesores.

La red tiene un aspecto como el siguiente:



Red de los ciclos del I.E.S. Gonzalo Nazareno

La idea consiste en que la conexión a Internet sea fluida incluso en casos de un uso intensivo.

En nuestro caso la máquina que nos interesa es `macaco` por el hecho de actuar como router de la red. Tiene aplicadas unas reglas de Iptables para el cortafuegos y ejecuta un servidor DHCP para ofrecer direcciones a las aulas. La interfaz en la que aplicaremos los

ajustes será *eth0*, que nos da salida a Internet estando conectada a un router en modo monopuesto. La línea contratada es un ADSL con Telefónica de 10Mbps/800Kbps.

3.1. Situación antes de la implantación

Como se ha comentado en la introducción del caso práctico, se ha optado por implantar un QoS para solventar los problemas que existen con respecto a la conexión ADSL compartida por tantos usuarios.

La navegación por páginas web es muy lenta cuando alguien está descargando a una velocidad muy elevada o subiendo del mismo modo. Es decir, no se puede trabajar cuando una sola persona es capaz de acaparar todo el ancho de banda.

A continuación, como una evidencia del problema, se puede mostrar la salida de un ping a una de las direcciones IP de Google cuando alguien sube un fichero a GMail por ejemplo.

```
spiki@nemola:~$ ping -c100 74.125.77.104
PING 74.125.77.104 (74.125.77.104) 56(84) bytes of data.
64 bytes from 74.125.77.104: icmp_seq=1 ttl=48 time=760 ms
64 bytes from 74.125.77.104: icmp_seq=2 ttl=48 time=800 ms
64 bytes from 74.125.77.104: icmp_seq=3 ttl=48 time=713 ms
64 bytes from 74.125.77.104: icmp_seq=4 ttl=48 time=752 ms
64 bytes from 74.125.77.104: icmp_seq=5 ttl=48 time=592 ms
       . . .
64 bytes from 74.125.77.104: icmp_seq=93 ttl=48 time=816 ms
64 bytes from 74.125.77.104: icmp_seq=94 ttl=48 time=941 ms
64 bytes from 74.125.77.104: icmp_seq=95 ttl=48 time=1031 ms
64 bytes from 74.125.77.104: icmp_seq=96 ttl=48 time=993 ms
64 bytes from 74.125.77.104: icmp_seq=97 ttl=48 time=1008 ms
64 bytes from 74.125.77.104: icmp_seq=98 ttl=48 time=1090 ms
64 bytes from 74.125.77.104: icmp_seq=99 ttl=48 time=1104 ms
64 bytes from 74.125.77.104: icmp_seq=100 ttl=48 time=1049 ms

--- 74.125.77.104 ping statistics ---
100 packets transmitted, 99 received, 1% packet loss, time 99615ms
rtt min/avg/max/mdev = 282.047/838.685/1549.602/206.056 ms, pipe 2
```

Como se puede observar, se han realizado 100 peticiones de ping a la IP 74.125.77.104 (Google) y mientras se subía el fichero se ha obtenido una media de 838ms de tiempo de respuesta.

3.2. ¿Qué es lo que vamos a hacer?

Lo que haremos será limitar nuestra velocidad de subida para evitar saturación en el módem (o router en monopuesto).
A esta operación se le llama *Shaping*.

Además, crearemos diferentes "bandas" de mayor a menor prioridad para los siguientes protocolos:

- ICMP, ACK, DNS.
- ToS Minimize Delay (por ejemplo SSH).
- HTTP.
- HTTPS y resto del tráfico.

Es decir, que siempre se desencolarán antes los paquetes con mayor prioridad. Además, cada "banda" tendrá un mínimo de ancho de banda garantizado y podrá "tomar prestado" todo el ancho de banda si es posible.
A esta última operación se le llama *Scheduling*.

Y para optimizar un poco más la situación se usará un algoritmo llamado SFQ (es una Qdisc) que hará posible que todas las conexiones (se puede interpretar como "todos los usuarios" aunque no es del todo cierto) tengan las mismas posibilidades de salir.

Además de la subida, se limitará de una forma un tanto peculiar la bajada.
Lo que haremos será descartar ciertos paquetes para que la velocidad de descarga se mantenga en un límite. Además ese límite irá a ráfagas, para que todos los equipos puedan tener cierta velocidad y posibilidades de descarga.
Esta acción se llama *Policing*.

Todo ésto se llevará a cabo con *TC* e *Iptables*, que explicaremos a continuación.

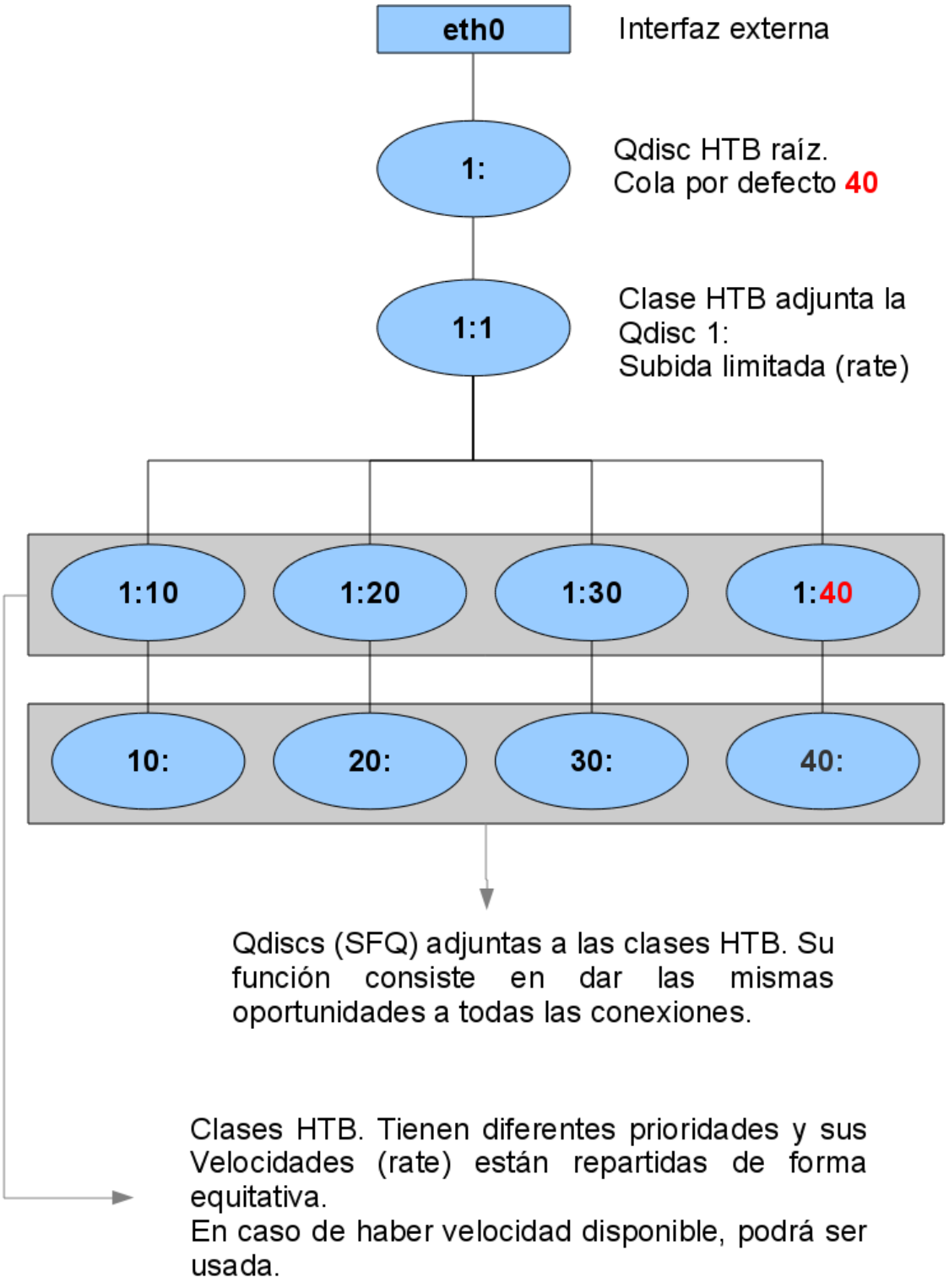
3.3. Disciplinas de cola (Qdiscs) usando TC

Las disciplinas de cola (qdiscs) son una serie de ajustes que se hacen sobre la interfaz de red que deseemos controlar para realizar QoS. Se crean los llamados árboles de preferencias usando Qdiscs, Qdiscs con clases, clases, etc. Cada uno de los elementos del árbol tendrá una serie de algoritmos y ajustes para controlar el tráfico. A grandes rasgos, estamos creando unos “canales” diferentes para los distintos tipos de tráfico que queramos controlar.

Todos estos conceptos se explicarán en el *Anexo de teoría* de este mismo documento.

Para explicar este ejemplo y tratar que sea más fácil de entender, hay que mencionar que todas la interfaces para las que hagamos un árbol de preferencias tienen una qdisc raíz de la que colgarán las demás qdiscs simples o con clases. A las qdiscs con clases se les pueden añadir qdiscs, qdiscs con clases o clases. A las clases se les pueden añadir también clases, qdiscs o qdiscs con clases. Así sucesivamente formando un árbol jerárquico.
Además, para saber qué elementos son los padres y los hijos en el árbol hay que darle un nombre a cada elemento (`handle` o `classid`), así como indicar cuál es su padre (`parent`).

El árbol de preferencias usado para nuestro caso práctico tiene un aspecto como el siguiente.



Árbol de preferencias (qdiscs y clases)

Se consigue formar la estructura anterior con las siguientes líneas.

```
# Interfaces definidas para establecer las preferencias
# y poder realizar el filtrado con Iptables.
IF_EXT=eth0
IF_INT=eth1

# Valores para la subida y la descarga de la línea.
SUBIDA="600kbit"       # Subida (algo menos de la real).
MEDIA="150kbit"        # La media para el rate de cada clase.
BAJADA="7.5mbit"       # Bajada (algo menos de la real).
BURST_BAJADA=".75m"    # Controlamos la ráfaga para el tráfico entrante.
COLA_DEFECTO="40"      # Cola por defecto para los paquetes sin marcar.

# Creamos la qdisc raiz y le agregamos la clase HTB para moldear el tráfico
tc qdisc add dev $IF_EXT root handle 1: htb default $COLA_DEFECTO
tc class add dev $IF_EXT parent 1: classid 1:1 htb rate $SUBIDA

# Añadimos las clases a la clase HTB para crear las bandas para cada tipo de
conexión.
tc class add dev $IF_EXT parent 1:1 classid 1:10 htb rate $MEDIA ceil $MEDIA
prio 1
tc class add dev $IF_EXT parent 1:1 classid 1:20 htb rate $MEDIA ceil $SUBIDA
prio 2
tc class add dev $IF_EXT parent 1:1 classid 1:30 htb rate $MEDIA ceil $SUBIDA
prio 3
tc class add dev $IF_EXT parent 1:1 classid 1:40 htb rate $MEDIA ceil $SUBIDA
prio 4

# Añadimos las Qdisc terminales. En nuestro caso SFQ.
# Con ello permitiremos que diferentes conexiones de cada cola tengan
# las mismas posibilidades de salir.
tc qdisc add dev $IF_EXT parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev $IF_EXT parent 1:20 handle 20: sfq perturb 10
tc qdisc add dev $IF_EXT parent 1:30 handle 30: sfq perturb 10
tc qdisc add dev $IF_EXT parent 1:40 handle 40: sfq perturb 10


# Clasificamos los paquetes marcados en las colas deseadas
tc filter add dev $IF_EXT protocol ip prio 1 parent 1:0 handle 1 fw classid
1:10
tc filter add dev $IF_EXT protocol ip prio 2 parent 1:0 handle 2 fw classid
1:20
tc filter add dev $IF_EXT protocol ip prio 3 parent 1:0 handle 3 fw classid
1:30
tc filter add dev $IF_EXT protocol ip prio 4 parent 1:0 handle 4 fw classid
1:40

# Controlamos la bajada (INGRESS)
tc qdisc add dev $IF_EXT handle ffff: ingress
tc filter add dev $IF_EXT parent ffff: protocol ip prio 50 u32 match ip src
0.0.0.0/0 police rate $BAJADA burst $BURST_BAJADA drop flowid :1
```

Explicación de las constantes definidas:

- `IF_EXT` e `IF_INT` son las interfaces pública e interna (aulas) respectivamente.
- `SUBIDA` y `BAJADA` establecen los límites (artificiales) de nuestra conexión. No se establecen al máximo teórico de la línea porque ésta no siempre puede tener esa capacidad y como consecuencia se crearían colas en el módem (o router en monopuesto), destruyendo así todo el trabajo que hemos realizado con las reglas QoS establecidas.

  Es cuestión de ir ajustando los parámetros de forma óptima a nuestra línea. Uno de los métodos más sencillos consiste en realizar pings mientras se hacen subidas para ver cómo están afectando los nuevos parámetros a nuestra conexión. Si los tiempos de respuesta empiezan a parecerse a los de una línea sin QoS, tiempos altos, habrá que disminuir un poco los valores. Sobre todo el de subida que es el más crítico.

  Se puede usar una herramienta como *Iperf* para averiguar cuál es nuestra subida real.
- `MEDIA` no es más que la media de nuestra velocidad de subida (constante SUBIDA) dividida entre las clases HTB que tengamos. En nuestro caso son 4 clases, por lo que la media es de 150Kbits/s.
- `COLA_DEFECTO` establece a qué clase irán los paquetes que no son marcados. En este caso es la 40, por lo que dichos paquetes tienen como destino la cola con menos prioridad.
- `BURST_BAJADA` suele establecerse como nuestra velocidad de bajada establecida dividida por 10. No hay mucha información sobre ésto pero funciona. Establece la ráfaga de descarga máxima.

Explicación de cada una de las líneas (TC):

```
tc qdisc add dev $IF_EXT root handle 1: htb default $COLA_DEFECTO
```

Establecemos la qdisc HTB raíz de la interfaz `IF_EXT` y establecemos la cola por defecto con el parámetro `default`.

```
tc class add dev $IF_EXT parent 1: classid 1:1 htb rate $SUBIDA
```

Añadimos una clase HTB adjunta a la qdisc raíz HTB. Limitamos la subida con el parámetro `rate`.

```
tc class add dev $IF_EXT parent 1:1 classid 1:10 htb rate $MEDIA ceil $MEDIA
prio 1
tc class add dev $IF_EXT parent 1:1 classid 1:20 htb rate $MEDIA ceil $SUBIDA
prio 2
tc class add dev $IF_EXT parent 1:1 classid 1:30 htb rate $MEDIA ceil $SUBIDA
prio 3
tc class add dev $IF_EXT parent 1:1 classid 1:40 htb rate $MEDIA ceil $SUBIDA
prio 4
```

Con las líneas anteriores creamos las 4 bandas que son en realidad 4 clases HTB con velocidad mínima asegurada (`rate`) igual a la constante MEDIA, la velocidad máxima de la que puede disponer si fuera posible (`ceil`) igual a la SUBIDA total establecida.
Además, con el parámetro `prio` establecemos las prioridades que tendrán dichas bandas, siendo los números menores los que otorgan mayor prioridad.

```
tc qdisc add dev $IF_EXT parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev $IF_EXT parent 1:20 handle 20: sfq perturb 10
tc qdisc add dev $IF_EXT parent 1:30 handle 30: sfq perturb 10
tc qdisc add dev $IF_EXT parent 1:40 handle 40: sfq perturb 10
```

A cada una de las clases HTB anteriores les adjuntamos unas qdiscs terminales SFQ. El parámetro `perturb` sirve para establecer el tiempo en el que se recalculará un hash interno de SFQ para que las conexiones se repartan de forma equitativa. Su valor recomendado es el establecido en nuestro caso y su magnitud son segundos.

```
tc filter add dev $IF_EXT protocol ip prio 1 parent 1:0 handle 1 fw classid
1:10
tc filter add dev $IF_EXT protocol ip prio 2 parent 1:0 handle 2 fw classid
1:20
tc filter add dev $IF_EXT protocol ip prio 3 parent 1:0 handle 3 fw classid
1:30
tc filter add dev $IF_EXT protocol ip prio 4 parent 1:0 handle 4 fw classid
1:40
```

Con estas líneas estableceremos la relación entre las marcas, que realizaremos con Iptables, y las qdiscs de nuestro árbol. Se entenderá mejor en el siguiente apartado.

```
tc qdisc add dev $IF_EXT handle ffff: ingress
tc filter add dev $IF_EXT parent ffff: protocol ip prio 50 u32 match ip src
0.0.0.0/0 police rate $BAJADA burst $BURST_BAJADA drop flowid :1
```

Con éstas últimas líneas establecemos la qdiscs para el tráfico de bajada (*ingress*). Los parámetros que se cambian son `rate` y `burst`. Lo demás se establece del mismo modo en cualquier implantación.

Para resumir, podemos decir que hemos creado una qdisc raíz HTB asociada a la interfaz de red, de la que cuelga una clase HTB con una tasa `rate` establecida. A su vez, de esa clase HTB cuelgan 4 clases que son nuestras bandas de diferentes prioridades, cada una de ellas con una velocidad mínima asegurada (`rate`), una máxima alcanzable (`ceil`) y su prioridad (`prio`). Finalizando, a cada una de esas clases les hemos asignado una qdisc SFQ para brindarle a todas las máquinas las mismas posibilidades de salir a Internet.

Una vez creado el árbol y agregados los filtros para colocar el tráfico en las diferentes colas, tan sólo falta establecer una marca a los paquetes que deseemos colocar en cada cola. Para ello usamos *Iptables*.

3.4. Marcado de paquetes con Iptables

Para marcar los paquetes con *Iptables* hacemos uso de la tabla *mangle* y establecemos dicha marca con la orden `MARK --set-mark *`. Las marcas son numéricas.

```
# Establecemos las reglas de Iptables para la tabla MANGLE
iptables -t mangle -A POSTROUTING -o $IF_EXT -p icmp -j MARK --set-mark 1
iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp –tcp-flags \
SYN,RST,ACK ACK -j MARK --set-mark 1
iptables -t mangle -A POSTROUTING -o $IF_EXT -p udp --dport 53 -j MARK \
--set-mark 1
iptables -t mangle -A POSTROUTING -o $IF_EXT -m tos --tos Minimize-Delay \
-j MARK --set-mark 2   # SSH
iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp --dport 80 -j MARK \
--set-mark 3
iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp --dport 443 -j MARK \
--set-mark 4
```

En este caso, las marcas que se han establecido son:

Marca	Tipo de paquete
1	ICMP, ACK, DNS
2	ToS (Minimize-Delay). SSH
3	HTTP (destino)
4	HTTPS (destino)

Con las líneas anteriores de Iptables tenemos clasificado el tráfico deseado. Además, atendiendo a la línea de la qdisc raíz en el apartado de TC, se puede observar como el tráfico por defecto (el no marcado) se envía directamente a la clase 1:40, la de menos prioridad.

Las marcas se relacionen con la estructura creada en TC mediante los filtros que definimos en el apartado anterior, con la directiva `handle * fw`. Donde `*` irá cada marca numérica establecida con Iptables.

Ya hemos conseguido por lo tanto priorizar el tráfico deseado y “hacer esperar” al restante.

3.5. QoS Lectivo y No-Lectivo. El Script

Con lo anterior se obtuvieron buenos resultados, pero sólo estábamos teniendo en cuenta las conexiones desde dentro hacia fuera. Es decir, si alguna máquina se quedaba encendida o simplemente por no haber contemplado los servicios http y https que se ofrecen al exterior, se obtenían muy malos resultados cuando se accedía desde fuera al servidor del departamento.

La solución al problema consistió en realizar dos scripts de QoS, uno para el horario de clase (*lectivo*) y otro para el resto (*no lectivo*).
Por lo tanto, se realizó un script en bash para automatizar y mantener fácilmente el QoS tanto *lectivo* como *no lectivo*.

3.5.1. Estructura del script

- **/etc/qosgn/***

  Son ficheros específicos para cada QoS, con sus Qdiscs y reglas de Iptables propias. En nuestro caso lo único que cambia es la marca establecida en Iptables. Donde * irá en cada caso *lectivo* o *nolectivo*.

- **/etc/qosgn/common**

  Este fichero contendrá las constantes usadas por los scripts de QoS. Es decir: nombres de interfaces de red, velocidades de subida y bajada, cola por defecto y ráfaga de bajada.

- **/etc/init.d/qosgn**

  Este es el script destinado a “demonizar” nuestro QoS (aunque no sea un demonio como tal). Los parámetros que posee para controlarlo son:

  - **start/stop**: Sirve para parar o arrancar el QoS. En el arranque, si no existe el fichero de control `/var/lib/cache/qosgn`, se iniciará el QoS para el horario lectivo por defecto y se escribirá `lectivo` en el fichero de control.
  - **restart**: Simplemente ejecuta un `stop` y luego un `start`.

- **lectivo/nolectivo**: Establece las reglas para el QoS indicado con el parámetro. Al establecerlo se escribirá `lectivo` o `nolectivo` en el fichero de control (`/var/lib/cache/qosgn`) según corresponda.
- **install/uninstall**: Agrega/elimina del directorio `/etc/cron.d` los ficheros `/etc/cron.d/qos_*` encargados de automatizar el QoS.
- **status**: Muestra el estado de las qdiscs, clases, filtros y reglas de Iptables aplicadas.

- **/etc/cron.d/qos_***

  Son unos ficheros que contienen una línea estilo crontab destinadas al lanzamiento del script `/etc/init.d/qosgn` de un modo determinado a la hora y día que corresponda.

- **/var/lib/cache/qosgn**

  Fichero de control necesario para el funcionamiento “con memoria” del parámetro `start` del script. Gracias a él se puede recodar el modo en el que estaba el script antes de pararlo, por lo que `start` arrancará el QoS en el modo que le indique ese fichero.

3.5.2. Ficheros del script

/etc/qosgn/lectivo

```
# Creamos la qdisc raiz y le agregamos la clase HTB
# para moldear el tráfico.
/sbin/tc qdisc add dev $IF_EXT root handle 1: htb default $COLA_DEFECTO
/sbin/tc class add dev $IF_EXT parent 1: classid 1:1 htb rate $SUBIDA

# Añadimos las clases a la clase HTB para crear las bandas
# para cada tipo de conexión.
/sbin/tc class add dev $IF_EXT parent 1:1 classid 1:10 htb rate \
$MEDIA ceil $MEDIA prio 1
/sbin/tc class add dev $IF_EXT parent 1:1 classid 1:20 htb rate \
$MEDIA ceil $SUBIDA prio 2
/sbin/tc class add dev $IF_EXT parent 1:1 classid 1:30 htb rate \
$MEDIA ceil $SUBIDA prio 3
/sbin/tc class add dev $IF_EXT parent 1:1 classid 1:40 htb rate \
$MEDIA ceil $SUBIDA prio 4
```

```
# Añadimos las Qdisc terminales. En nuestro caso SFQ.
# Con ello permitiremos que diferentes conexiones de
# cada cola tengan las mismas posibilidades de salir.
/sbin/tc qdisc add dev $IF_EXT parent 1:10 handle 10: sfq perturb 10
/sbin/tc qdisc add dev $IF_EXT parent 1:20 handle 20: sfq perturb 10
/sbin/tc qdisc add dev $IF_EXT parent 1:30 handle 30: sfq perturb 10
/sbin/tc qdisc add dev $IF_EXT parent 1:40 handle 40: sfq perturb 10

# Establecemos las reglas de Iptables para la cadena POSTROUTING
# de la tabla MANGLE.
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p icmp \
-j MARK --set-mark 1
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp \
--tcp-flags SYN,RST,ACK ACK -j MARK --set-mark 1
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p udp \
--dport 53 -j MARK --set-mark 1
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -m tos\
--tos Minimize-Delay -j MARK --set-mark 2
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp\
--dport 80 -j MARK --set-mark 3
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp\
--dport 443 -j MARK --set-mark 4

# Filtros. Asociamos las Qdiscs con las marcas de Iptables.
/sbin/tc filter add dev $IF_EXT protocol ip prio 1 parent 1:0 \
handle 1 fw classid 1:10
/sbin/tc filter add dev $IF_EXT protocol ip prio 2 parent 1:0 \
handle 2 fw classid 1:20
/sbin/tc filter add dev $IF_EXT protocol ip prio 3 parent 1:0 \
handle 3 fw classid 1:30
/sbin/tc filter add dev $IF_EXT protocol ip prio 4 parent 1:0 \
handle 4 fw classid 1:40

# Controlamos la bajada (INGRESS)
/sbin/tc qdisc add dev $IF_EXT handle ffff: ingress
/sbin/tc filter add dev $IF_EXT parent ffff: protocol ip prio 50 \
u32 match ip src 0.0.0.0/0 police rate $BAJADA burst $BURST_BAJADA \
drop flowid :1
```

El fichero `/etc/qosgn/nolectivo` tan sólo tiene diferentes sus líneas de Iptables:

Líneas de Iptables del fichero /etc/qosgn/nolectivo

```
# Establecemos las reglas de Iptables para la cadena POSTROUTING
# de la tabla MANGLE.
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p icmp \
-j MARK --set-mark 1
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp \
--tcp-flags SYN,RST,ACK ACK -j MARK --set-mark 1
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p udp \
--dport 53 -j MARK --set-mark 1
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -m tos \
--tos Minimize-Delay -j MARK --set-mark 2
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp \
--sport 80 -j MARK --set-mark 3
/sbin/iptables -t mangle -A POSTROUTING -o $IF_EXT -p tcp \
--sport 443 -j MARK --set-mark 4
```

/etc/qosgn/common

```
# Interfaces definidas para establecer las prefrencias
# y poder realizar el filtrado con Iptables.

IF_EXT=eth0
IF_INT=eth1


# Valores para la subida y la descarga de la línea.

SUBIDA="600kbit"  # Subida (algo menos de la real).
MEDIA="150kbit"         # La media para el rate de cada clase.
BAJADA="7.5mbit"  # Bajada (algo menos de la real).
BURST_BAJADA=".75m"     # Controlamos la ráfaga para la bajada.
COLA_DEFECTO="40" # Donde irán los paquetes no marcados.
```

/etc/cron.d/qos_lectivo

```
0 8 * * 1-5 root  /etc/init.d/qosgn lectivo 2> /dev/null
```

/etc/cron.d/qos_nolectivo

```
0 15 * * 1-5      root  /etc/init.d/qosgn nolectivo 2> /dev/null
```

/etc/init.d/qosgn

```
#!/bin/bash

### BEGIN INIT INFO
# Provides:          qosgn
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Script de control del QoS
# Description:       Este fichero te ayudará a manejar el QoS
#                 para la infraestructura del IES GN.
### END INIT INFO

# Cargamos las variables que usaremos a lo largo del script
source /etc/qosgn/common

# Función encargada de borrar los ajustes de QoS
# de la interfaz IF_EXT
function borrar_ajustes
{
   /sbin/tc qdisc del dev $1 root 2>/dev/null>/dev/null
   /sbin/tc qdisc del dev $1 ingress 2>/dev/null>/dev/null
   /sbin/iptables -t mangle -F
}


# Controlamos los parámetros del script

case "$1" in
  start)
    # Arranque del script. Se usará, si existe el fichero
```

```
  # /var/lib/cache/qosgn. Si no existe se creará.
  if [ -e /var/lib/cache/qosgn ]
  then
      if [ `cat /var/lib/cache/qosgn` != "lectivo" ]
      then
          /etc/init.d/qosgn nolectivo
      else
          /etc/init.d/qosgn lectivo
      fi
  else
      /etc/init.d/qosgn lectivo
  fi
  ;;

stop)
  echo "** Parando el QoS. Se establecen las qdiscs por defecto."
  borrar_ajustes $IF_EXT
  ;;

restart)
  # Se reinician los ajustes.
  /etc/init.d/qosgn stop
  /etc/init.d/qosgn start
  ;;

install)
  echo "** Agregando el QoS al Cron "
  echo "0 8 * * 1-5   root  /etc/init.d/qosgn lectivo \
    2> /dev/null" > /etc/cron.d/qos_lectivo
  echo "0 15 * * 1-5  root  /etc/init.d/qosgn nolectivo \
    2> /dev/null" > /etc/cron.d/qos_nolectivo
  ;;

uninstall)
  echo "** Quitando el QoS del Cron "
  rm -f /etc/cron.d/qos_*
  ;;

lectivo)
  echo "** Aplicando QoS para horario lectivo "
  borrar_ajustes $IF_EXT
  source /etc/qosgn/lectivo
  echo "lectivo" > /var/lib/cache/qosgn
  ;;

nolectivo)
  echo "** Aplicando QoS para horario no lectivo "
  borrar_ajustes $IF_EXT
  source /etc/qosgn/nolectivo
  echo "nolectivo" > /var/lib/cache/qosgn
  ;;

status)
  echo -e "** Mostrando ajustes de la interfaz '$IF_EXT'\n"
  /sbin/tc qdisc show dev $IF_EXT
  echo -e "\n"
  /sbin/tc class show dev $IF_EXT
  echo -e "\n"
  /sbin/tc filter show dev $IF_EXT
```

```
    echo -e "\n** Mostrando filtros en Iptables (mangle)"
    /sbin/iptables -t mangle -vnL POSTROUTING
    ;;

  *)
    echo "Uso: /etc/init.d/qosgn {start|stop|restart|install| \
      uninstall|lectivo|nolectivo|status}"
    exit 1
    ;;

esac

exit 0
```

3.6. Debianizando el script

Los ficheros creados anteriormente, una vez establecidos en sus directorios correspondientes, hacen que podamos controlar nuestro QoS de una forma sencilla.

Pero, para facilitar la instalación/desinstalación del script y dar más sensación de limpieza, se ha optado por crear un paquete *.deb*.

Para ello, habrá que explicar qué es en realidad un paquete *.deb* y cómo podemos construir uno de una forma sencilla y sin utilizar aplicaciones adicionales.

3.6.1. Qué es y cómo se estructura un paquete DEB

Un paquete *.deb* no es más que un fichero en formato *ar* estándar que contiene un fichero en texto plano y dos tarballs (tar, tar.gz, tar.bz2 o lzma).

Con lo cual, colocando los ficheros en los tarballs correspondientes y empaquetándolo todo obtendremos nuestro paquete *.deb*.

La estructura interna es la siguiente:

- **debian-binary**: Es un fichero en texto plano que contiene la versión del formato *DEB*. En la actualidad se establece como `2.0`.
- **control.tar.gz**: Todos los ficheros que contiene son en texto plano. Entre otros, contiene un fichero llamado control que establece el nombre del paquete, dependencias del mismo, versión, descripción, paquetes recomendados, nombre del mantenedor, etc.

Por otra parte, contiene scripts (normalmente en bash) con nombres del tipo `postrm`, `preinst`, `postinst`, `prerm`, etc. Ayudarán a automatizar tareas en diferentes puntos de la instalación/desinstalación/reconfiguración del paquete. Dependiendo de qué se necesite se usará uno u otro, o una combinación de ellos.
Cabe destacar que los ficheros comprimidos en el `control.tar.gz` no tienen ninguna ruta, se comprimen sin ningún tipo de directorio.

- **data.tar.gz**: Contiene todos los ficheros de funcionamiento del programa o en nuestro caso los del QoS. Deben estar estructurados tal y como quedarán en el sistema.

Una vez empaquetados los ficheros de control y de datos quedaría empaquetar con *ar* todos esos ficheros y nombrar el paquete con extensión *.deb*.

3.6.2. Creación de los ficheros de control

Empezaremos por el fichero control.

control

```
Package: qosgn
Architecture: all
Version: 0.1
Maintainer: Carlos Álvarez <caralvmar@gmail.com>
Section: net
Priority: extra
Homepage: http://informatica.gonzalonazareno.org/
Depends: iproute, iptables
Installed-Size: 19
Description: Scripts para hacer QoS en el I.E.S. GN
 Contiene scripts para la gestión automática
 del QoS en la infraestructura del departamento
 de informática del I.E.S. Gonzalo Nazareno.
```

Esos son los metadatos de nuestro futuro paquete *.deb*. Se pueden observar entre otros el nombre del paquete (`Package`), su versión, el mantenedor, la sección, dependencias, etc.

Cabe mencionar que la descripción en realidad son dos partes. La línea que empieza con `Description` es independiente a la parte de abajo. Lo primero es la descripción corta y el resto es la larga. La larga tiene que empezar por un espacio en blanco.

Seguimos creando los scripts de control. En nuestro caso serán `postinst` y `prerm`. Es decir, tan sólo realizaremos acciones extra justo después de instalar el paquete y antes de desinstalarlo.

postinst

```
#!/bin/sh
mkdir -p "/var/lib/cache/"
/etc/init.d/qosgn install
/etc/init.d/qosgn start
update-rc.d qosgn defaults
```

Justo después de instalar el paquete crearemos el directorio para nuestro fichero de control del script. Posteriormente instalaremos el script en el Cron, lo arrancaremos y por último estableceremos los enlaces simbólicos para los diferentes niveles de ejecución.

prerm

```
#!/bin/sh
/etc/init.d/qosgn stop
/etc/init.d/qosgn uninstall
update-rc.d -f qosgn remove
```

En este caso, lo que hacemos antes de desinstalar el paquete es parar el QoS, quitarlo del Cron y finalmente eliminar los enlaces simbólicos de los directorios `/etc/rc*.d/` .

3.6.3. Creación del paquete DEB

Para crear el paquete *.deb* seguiremos los siguientes pasos, estando en un directorio con los ficheros del script organizados en diferentes subdirectorios (`datos` y `control`):

Lo primero que tenemos que hacer es trabajar como root y darle permisos para tal usuario:

```
nemola:~/script# chown -R root:root *
nemola:~/script# chmod -R 640 *
nemola:~/script# ls -l
total 12
drw-r----- 2 root root 4096 jun  1 19:30 control
drw-r----- 3 root root 4096 jun  1 19:31 datos
-rw-r----- 1 root root    4 jun  1 18:50 debian-binary
```

Entramos en `datos/` y creamos el tarball del directorio que hay en su interior:

```
nemola:~/script# cd datos/
nemola:~/script/datos# ls -l
total 4
drw-r----- 4 root root 4096 jun  1 18:50 etc
```

```
nemola:~/script/datos# tar -czvf data.tar.gz ./
./
./etc/
./etc/qosgn/
./etc/qosgn/common
./etc/qosgn/lectivo
./etc/qosgn/nolectivo
./etc/init.d/
./etc/init.d/qosgn
```

```
nemola:~/script/datos# mv data.tar.gz ../
```

Ahora, nos situamos en `control/` y pasamos a empaquetar nuestro ficheros:

```
nemola:~/script/datos# cd ../control/
nemola:~/script/control# ls -l
total 12
-rw-r----- 1 root root 414 jun  1 19:21 control
-rw-r----- 1 root root 114 jun  1 18:50 postinst
-rw-r----- 1 root root  89 jun  1 18:50 prerm
```

```
nemola:~/script/control# tar -czvf control.tar.gz ./
./
./prerm
./postinst
./control
tar: .: el fichero cambió mientras se estaba leyendo
```

```
nemola:~/script/control# mv control.tar.gz ../
```

Sólo nos queda situarnos en la raíz de nuestro directorio de trabajo (`script/`) donde ya tendremos los tarballs de datos y control, además de nuestro fichero `debian-binary` que ya teníamos creado previamente. Tan sólo contiene `2.0` .

Pasamos a la creación del paquete *.deb*. Hay que usar el comando *ar* para crear el archivador introduciendo en él los ficheros anteriores en el siguiente orden (si no, el paquete no será reconocido por *dpkg*):

1. `debian-binary`
2. `control.tar.gz`
3. `data.tar.gz`

Para crearlo tendríamos que ejecutar el siguiente comando:

```
nemola:~/script/datos# cd ../
nemola:~/script# ar -rc qosgn_0.1_all.deb debian-binary \
control.tar.gz data.tar.gz
```

Si listamos el contenido del directorio veremos que ya tenemos el paquete *.deb* listo para usar.

```
nemola:~/script# ls -l
total 24
drw-r----- 2 root root 4096 jun 19 01:34 control
-rw-r--r-- 1 root root  579 jun 19 01:34 control.tar.gz
-rw-r--r-- 1 root root 2083 jun 19 01:30 data.tar.gz
drw-r----- 3 root root 4096 jun 19 01:31 datos
-rw-r----- 1 root root    4 jun  1 18:50 debian-binary
-rw-r--r-- 1 root root 2856 jun 19 01:40 qosgn_0.1_all.deb
```

A la hora de nombrar el paquete se ha seguido el estándar de Debian.

nombre_versión_arquitectura.deb

3.7. MRTG: Monitorización de la interfaz externa

Usaremos la aplicación *MRTG* (Multi Router Traffic Grapher) para monitorizar el tráfico de subida y bajada de la interfaz `eth0` de `macaco`.

Ésto nos ayudará a detectar si nuestra línea se encuentra saturada en algún momento y con lo cual poder tomar medidas para solventarlo, ya sea modificando el script de QoS, contratando más ancho de banda, restringiendo el uso de la red, etc.

La única dependencia que necesitamos es un servidor web donde poder mostrar las gráficas. Para nuestro caso usaremos *apache2*, por lo que procedemos a instalarlo con:

```
macaco:~# aptitude install apache2
```

A continuación instalamos *MRTG*, como no, usando aptitude:

```
macaco:~# aptitude install mrtg mrtg-contrib mrtgutils
```

Ahora tan sólo quedaría modificar/agregar unas líneas en el fichero de configuración

`/etc/mrtg.cfg` para que quede como a continuación:

/etc/mrtg.cfg

```
# Global configuration
WorkDir: /var/www/mrtg
Language: spanish
WriteExpires: Yes
Options[_]: growright

Target[eth0]: `/usr/bin/mrtg-ip-acct eth0`
MaxBytes1[eth0]: 1200000
MaxBytes2[eth0]: 100000
Title[eth0]: Analisis de trafico total para eth0
YLegend[eth0]: Trafico
PageTop[eth0]: <h2>Analisis del trafico total</h2>
XScale[eth0]: 2.5
YScale[eth0]: 2.5
```

En el fichero hemos establecido el directorio de trabajo (donde se guardarán las gráficas y los logs de funcionamiento), el lenguaje, autorefresco de la página html (`WriteExpires`), crecimiento de la gráfica de derecha a izquierda, la interfaz de donde capturaremos los datos, los Bytes máximos tanto para la bajada (`MaxBytes1`) como la subida (`MaxBytes2`), el título de la página html, la leyenda de Y, la cabecera de la página y la escala X e Y de la gráfica representada.
Con ésta ultima escala conseguimos que pueda apreciarse algo más la subida, puesto que al tener una línea tan asimétrica la subida (menos de 1/10 parte de la bajada) es casi despreciable .

Una vez guardado el fichero de configuración hay que crear el directorio de trabajo `/var/www/mrtg` y ejecutar MRTG por primera vez para la generación inicial de los ficheros.

```
macaco:~# mkdir /var/www/mrtg
macaco:~# env LANG=C /usr/bin/mrtg /etc/mrtg.cfg
```

Por otra parte, para acceder directamente a las estadísticas de nuestra interfaz externa hay que crear un enlace simbólico del fichero `eth0.html` como `index.html` y añadir un nuevo sitio en apache para MRTG.

```
macaco:~# ln -s /var/www/mrtg/eth0.html /var/www/mrtg/index.html
macaco:~# touch /etc/apache2/sites-available/mrtg
```

Modificamos el fichero `/etc/apache2/sites-available/mrtg`:

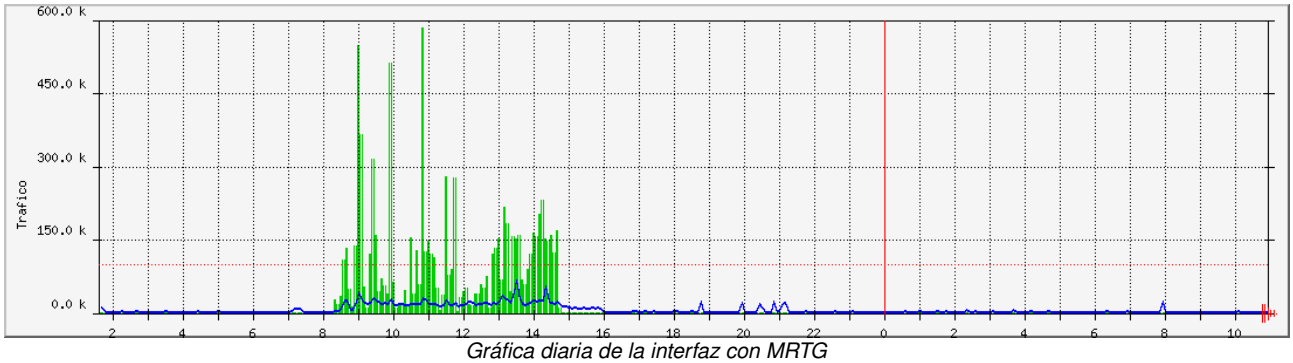
/etc/apache2/sites-available/mrtg

```
<VirtualHost *:80>
      ServerAdmin webmaster@localhost

      DocumentRoot /var/www/mrtg/
      <Directory />
            Options FollowSymLinks
            AllowOverride None
      </Directory>
      <Directory /var/www/mrtg/>
            Options Indexes FollowSymLinks MultiViews
            AllowOverride None
            Order allow,deny
            allow from all
      </Directory>

      ErrorLog /var/log/apache2/error.log

      # Possible values include: debug, info, notice, warn, error,
      # crit, alert, emerg.
      LogLevel warn

      CustomLog /var/log/apache2/access.log combined

</VirtualHost>
```

Una vez que hayamos salvado el fichero, habilitamos el nuevo sitio, deshabilitamos el sitio por defecto y recargamos *apache*.

```
macaco:~# a2ensite mrtg
macaco:~# a2dissite default
macaco:~# /etc/init.d/apache2 reload
```

A partir de ahora podremos ver las estadísticas (actualizadas cada 5 minutos) de nuestra interfaz de red externa desde http://macaco.



Gráfica diaria de la interfaz con MRTG

3.8. Medición del ancho de banda con Iperf

Se usó esta herramienta extremadamente sencilla pero a la vez útil. Opino que resultaba conveniente mencionarla dado que es importante saber qué velocidades reales puede alcanzar nuestra conexión para establecer adecuadamente nuestras constantes en el script de QoS.

Para la instalación del programa en distribuciones Debian:

```
macaco:~# aptitude install iperf
```

Para medir la velocidad de subida de macaco (de nuestra conexión) habría que lanzar en un extremo en Internet *Iperf* como servidor y en `macaco` lanzarlo como cliente conectando con dicho servidor.

Servidor:

```
unamaquina:~# iperf -s
```

Cliente:

```
macaco:~# iperf -c unamaquina.dominio.com
```

Ahora, en ambos extremos, se mostrará la velocidad a la que ha sido capaz de transmitir `macaco`. O lo que es lo mismo, obtendremos la velocidad de subida de nuestra línea en ese momento.

3.9. Situación después de la implantación

Una vez implantado el QoS, se estuvo probando varios días con bastante actividad en la red y se comportó de forma correcta incluso notándose una mejoría en la velocidad de navegación por ejemplo.

Para corroborarlo “sobre el papel” , volveremos a realizar la misma prueba de hacer pings a una IP de Google mietras subimos el mismo fichero que en el caso inicial a GMail. La IP es la misma y suele ser bastante estable, con lo que los resultados son bastante seguros.

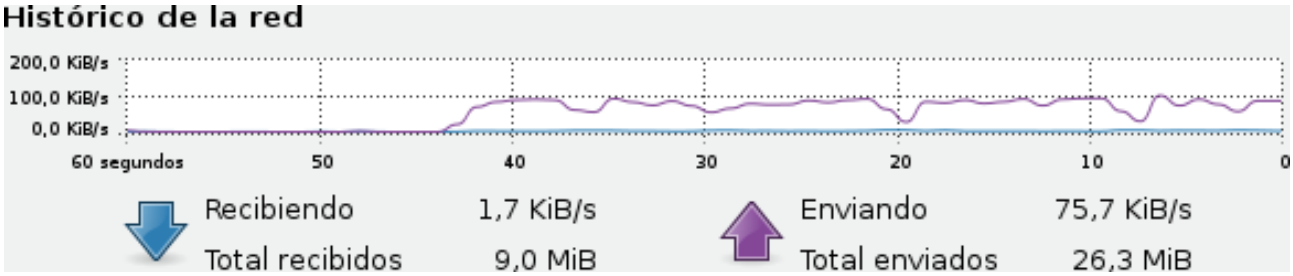
```
spiki@nemola:~$ ping -c100 74.125.77.104
PING 74.125.77.104 (74.125.77.104) 56(84) bytes of data.
64 bytes from 74.125.77.104: icmp_seq=1 ttl=48 time=95.1 ms
64 bytes from 74.125.77.104: icmp_seq=2 ttl=48 time=100 ms
64 bytes from 74.125.77.104: icmp_seq=3 ttl=48 time=90.5 ms
64 bytes from 74.125.77.104: icmp_seq=4 ttl=48 time=111 ms
64 bytes from 74.125.77.104: icmp_seq=5 ttl=48 time=211 ms
...
64 bytes from 74.125.77.104: icmp_seq=93 ttl=48 time=145 ms
64 bytes from 74.125.77.104: icmp_seq=94 ttl=48 time=151 ms
64 bytes from 74.125.77.104: icmp_seq=95 ttl=48 time=95.9 ms
64 bytes from 74.125.77.104: icmp_seq=96 ttl=48 time=90.2 ms
64 bytes from 74.125.77.104: icmp_seq=97 ttl=48 time=199 ms
64 bytes from 74.125.77.104: icmp_seq=98 ttl=48 time=157 ms
64 bytes from 74.125.77.104: icmp_seq=99 ttl=48 time=257 ms
64 bytes from 74.125.77.104: icmp_seq=100 ttl=48 time=234 ms

--- 74.125.77.104 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99727ms
rtt min/avg/max/mdev = 90.251/153.197/300.089/57.881 ms
```

Se puede observar cómo hemos pasado de tener un ping de ~800ms a ~150ms.

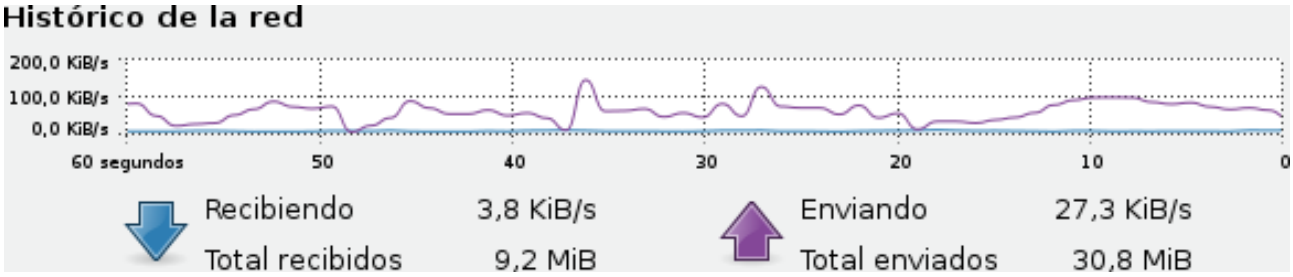
En las siguientes gráficas se puede comprobar además que cuando no está activo el QoS, la representación de la subida es bastante regular, lo que quiere decir que se está saturando la línea con una sola conexión.
Sin embargo, cuando se activa el QoS se observa una gráfica mucho más irregular, por lo que la subida no se satura. Es decir, cuando otra máquina u otra conexión intenta salir a

Internet, automáticamente nuestras qdiscs bajan la transferencia de subida y permiten más conexiones.

Las gráficas representan en ambos casos un fragmento de la subida de un fichero a GMail, que para nuestro caso es el tráfico con menos prioridad puesto que es HTTPS y se envía a la última cola.



Subida a GMail sin QoS



Subida a GMail con QoS

4. Anexo de Teoría

A continuación se exponen explicaciones con un poco de más profundidad acerca de los elementos con los que realizamos QoS. Se empezará definiendo la terminología usada.

4.1. Terminología

1. **Qdisc:** Traducido al castellano significa disciplina de cola (Queueing Discipline) y no son más que una serie de algoritmos que controlan de una determinada forma la cola de una interfaz de red. A priori, se puede aplicar tanto para la entrada de la interfaz (*ingress*) como para la salida (*egress*).
2. **Qdisc raíz (root):** Es aquella qdisc que se aplica de forma directa a la interfaz. De ella, dependiendo del tipo de qdisc que sea, podrán colgar otra serie de qdiscs.
3. **Qdisc sin clases:** Son qdiscs a las que no se pueden añadir subdivisiones internas. Son las más básicas. Un ejemplo podría ser una cola 'fifo' clásica.
4. **Qdisc con clases:** Son qdiscs, como las anteriores, pero que pueden contener clases. Las clases podrán contener a su vez más qdiscs con clases o clases.
5. **Clases:** Las clases son una serie de divisiones que se realizan sobre qdiscs con clases o sobre las propias clases. Las clases, qdisc y qdisc con clases mantienen relaciones padre/hijo. Al crear una clase se le añade una qdisc fifo por defecto (en concreto la pfifo_fast). Si a esa clase no se le añaden más clases o qdisc con clases se dice que es una qdisc terminal. La qdisc por defecto se puede sustituir por la que se desee en función de las necesidades que se tengan. Si a esa clase terminal le añadiésemos otra clase dejaría de ser una clase terminal y por tanto desaparecer su qdisc adjunta.
6. **Clasificador:** Es el mecanismo por el cual una qdisc con clases sabe a qué qdisc enviar el paquete en el momento de procesarlo.
7. **Filtro:** Es donde realmente se hace la clasificación de los paquetes. Este concepto y el anterior están muy relacionados y no tendría mucho sentido usar uno de ellos sin el otro. Se pueden usar diversos métodos, pero en esta memoria nos centraremos en usar el parámetro filter de la herramienta TC e IPTABLES como

clasificador.

8. **Scheduling:** También conocido como ordenamiento. Se trata de aplicar un determinado orden a la hora de poner los paquetes en una cola antes de ser pasados al driver de la tarjeta de red. Todo ello se consigue aplicando clasificadores y filtros.
9. **Shaping:** O ajuste, es el proceso por el cual retrasamos los paquetes en la salida de la interfaz (egress). Con ello conseguimos establecer un límite deseado en la interfaz en la salida en lugar de usar el límite físico disponible (el que permita el driver/hardware/enlace).
10. **Policing:** Es una técnica usada para ajustar el ancho de banda disponible para la cola de entrada de la interfaz (ingress). Lo que se hace en esta fase es descartar paquetes para que la velocidad del tráfico quede reducida a la deseada.

    Nota: Es justo lo contrario que realiza un ISP a la hora de ofrecernos la capacidad de descarga. Como es un límite que pone la calidad del enlace, el ISP siempre empieza con descargas muy rápidas (colas muy grandes) conforme se van descartando paquetes y ajustándose a la velocidad de la línea (enlace).

4.2. Un poco más antes de empezar

El tráfico que podemos controlar a la hora de realizar QoS es el tráfico saliente (egress). Es decir, el que se genera en nuestro kérnel (ya sea haciendo forward a otras máquinas o generándolo localmente).

El tráfico de entrada (ingress), sin embargo, no está a nuestro alcance al cien por cien. No podemos crear colas en el lado del ISP para controlar cuánto tráfico nos llega a nuestra interfaz. Poniendo un ejemplo que he visto en varios manuales se dice que: “En el correo postal no podemos controlar cuantas cartas llegan a la oficina de correo”. Aún así, no está todo perdido y existe una qdisc especial que puede coexistir con la qdisc raíz de nuestra interfaz que controlará en cierta medida la cola ingress. Es un mecanismo que realiza “policing”. Es decir, descarta paquetes para que paulatinamente vayan entrando a una

velocidad establecida.

Concluyendo:

Tráfico egress (salida)	“Control total”.
Tráfico ingress (entrada)	“Cierto control de forma limitada”.

4.3. Algunas disciplinas de cola (Qdiscs)

Aquí trataremos de explicar algunas de las qdiscs más usadas. Ya sean sin clases o con clases.

4.3.1. Qdiscs sin clases

Como dijimos anteriormente son las qdiscs a las que no se le pueden añadir clases. Es decir, no podemos hacer clasificaciones internas a ellas. Entre otras muchas, existen las siguientes:

4.3.1.1. pfifo_fast

Es la qdisc que se asigna por defecto a una interfaz en el núcleo Linux normalmente (un núcleo para “Advanced routing”). También es la qdisc que se asigna a una clase recién creada y a la que no se ha añadido ninguna qdisc personalizada o ninguna otra clase.
Esta qdisc es un poco peculiar puesto que no se pueden cambiar sus ajustes mediante la herramienta TC. Todo se realizará desde 'ifconfig'. Además, a diferencia de una cola FIFO tradicional, internamente trabaja con una serie de bandas, por lo que a priori se podría decir que se trata de una qdisc con clases.
Ahora la pregunta es que si tiene una serie de bandas y no se puede manejar con TC, ¿cómo saben los paquetes a qué banda deben ir?
La respuesta es: Atendiendo al campo TOS (Type of Service) de una trama IP.

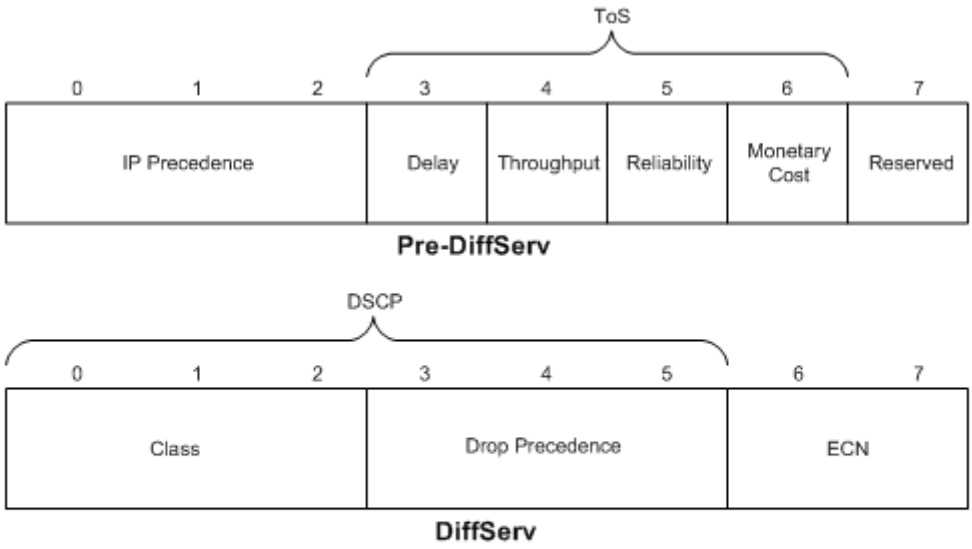
La qdisc pfifo_fast y el núcleo Linux, tienen por defecto un mapeo que indicará, dependiendo del campo TOS del paquete, a qué banda de la qdisc irá dicho paquete. Este mapeo se llama PRIOMAP, y por defecto realiza las siguientes asignaciones.
(Ver imagen “Priomap en Linux” en la siguiente página).

Por ejemplo, SSH establece un valor en el campo TOS (0x10) para que al usar este tipo de colas, el paquete sufra el menor retraso posible.

TOS	Bits	Significa	Prioridad Linux	Banda
0x0	0	Servicio normal	0 Mejor esfuerzo	1
0x2	1	Minimizar coste monet.	1 Relleno	2
0x4	2	Maximizar fiabilidad	0 Mejor esfuerzo	1
0x6	3	mmc+mr	0 Mejor esfuerzo	1
0x8	4	Mazimizar transferencia	2 En masa	2
0xa	5	mmc+mt	2 En masa	2
0xc	6	mr+mt	2 En masa	2
0xe	7	mmc+mr+mt	2 En masa	2
0x10	8	Minimizar retrasos	6 Interactivo	0
0x12	9	mmc+md	6 Interactivo	0
0x14	10	mr+md	6 Interactivo	0
0x16	11	mmc+mr+md	6 Interactivo	0
0x18	12	mt+md	4 Int. en masa	1
0x1a	13	mmc+mt+md	4 Int. en masa	1
0x1c	14	mr+mt+md	4 Int. en masa	1
0x1e	15	mmc+mr+mt+md	4 Int. en masa	1

Priomap en Linux

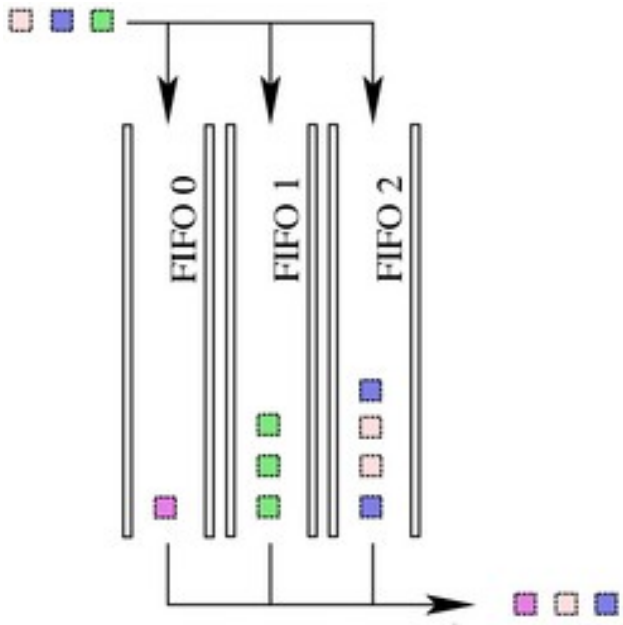
A modo de refrescar un poco la memoria, se puede observar con la siguiente imagen el aspecto del campo TOS de un datagrama IP.



IP ToS Byte Standards

Para concluir con la explicación de pfifo_fast, se puede ver reflejado su funcionamiento en

la siguiente imagen:



Se puede obtener más ayuda con '$man tc-pfifo_fast'.

4.3.1.2. TBF (Token Bucket Filter)

Esta qdisc tiene la finalidad de limitar el tráfico que pasará por la interfaz (estamos realizando 'shaping' tan solo). Se impone una tasa específica a la que pasarán los paquetes además de un parámetro 'burst' (ráfaga) que nos permitirá indicar el valor máximo al que podrá transmitir la interfaz por encima del límite. Esto último es útil para paquetes o transmisiones pequeñas.

En la siguiente página se puede ver una imagen que ayuda a comprender el funcionamiento de esta qdisc:

Se puede obtener más ayuda con '$man tc-tbf'.

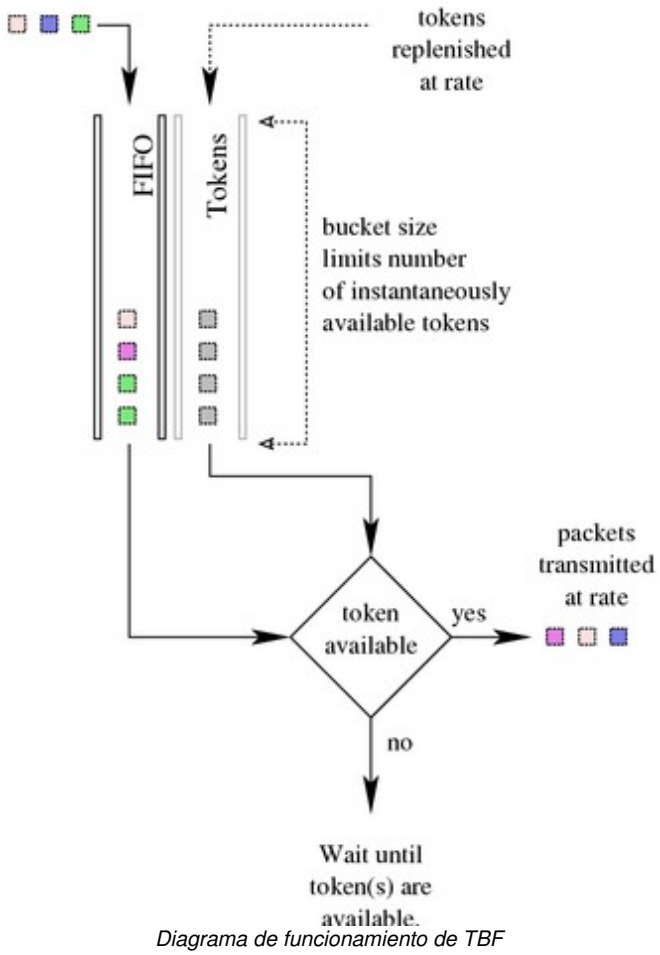


Diagrama de funcionamiento de TBF

4.3.1.3. SFQ (Stochastic Fairness Queueing)

Es una qdisc que no tiene mucho sentido si no se utiliza bajo una qdisc con clases en el caso de que la interfaz a la que la vayamos a aplicar tenga la misma transferencia que el enlace (por ejemplo un modem). Esto es porque no hace ningún tipo de ajuste (shaping) al tráfico, y aunque se reordene el tráfico (scheduling) existirán colas y la interactividad se verá afectada.

SFQ intenta que el flujo de datos sea justo para todas las conexiones (que no tienen que provenir de máquinas diferentes) y tengan las mismas posibilidades de salir.

Crea una serie de FIFOs internas y mediante un algoritmo de hash se las asigna a cada conexión.

Para que no haya coincidencias a la hora de las prioridades de cada conexión existe un parámetro ('perturb') que indica cada cuanto tiempo se recalcula el hash. (Suele ponerse

cada 10 segundos).

Finalmente, a esas FIFOs se les da salida de acuerdo a un algoritmo RoundRobin.

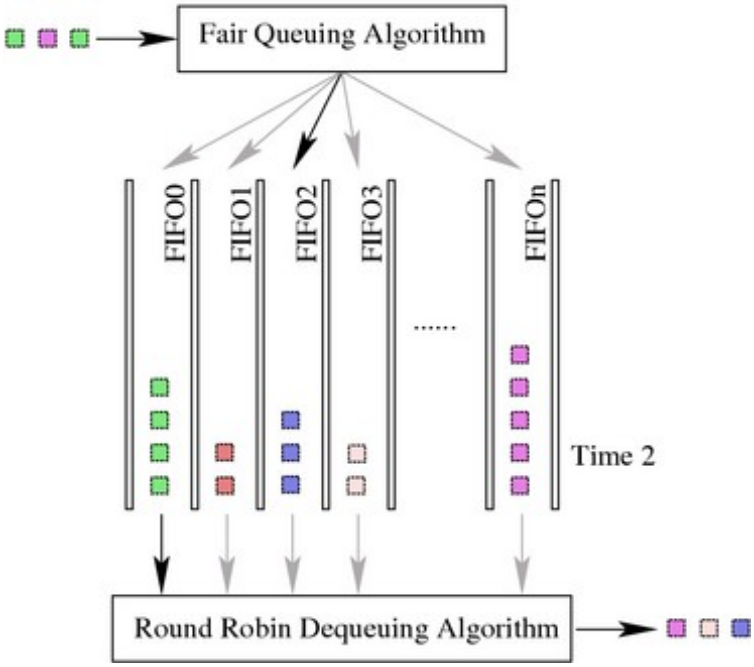


Diagrama de funcionamiento de SFQ

Es interesante mencionar que existe una versión actualizada, ESFQ (Enhanced Stochastic Fairness Queueing), que permite ajustar el tráfico por dirección IP y no por conexión. A día de hoy se puede usar esta qdisc parcheando y recompilando el núcleo Linux, con lo que puede no ser una solución viable para un sistema en producción. El parche en cuestión puede encontrarse en http://fatooh.org/esfq-2.6/ .

Normalmente, SFQ se suele usar estableciendo tan sólo el parámetro 'perturb', pero se puede establecer también la cantidad de bytes que deberá soltar cada cola FIFO de SFQ antes de pasar a la siguiente (parámetro 'quantum'). El parámetro 'quantum' nunca debe ser menor que el MTU de la interfaz, que es el predeterminado. Además, se recomienda dejar el valor por defecto.

Se puede obtener más información con '$man tc-sfq'.

4.3.2. Qdiscs con clases

Las qdiscs con clases son colas como las anteriores, pero se le pueden añadir una serie de bandas o mejor dicho, clases. Cada clase tendrá una característica, como por ejemplo un tasa mínima de velocidad garantizada ('rate'). De modo que, con las qdiscs con clases realizaremos shaping.

Gracias a las qdisc con clases se pueden crear los llamados árboles, que no son más que un agrupamiento de qdisc con clases y sin clases, organizadas de forma jerárquica, por las que pasarán los paquetes.

4.3.2.1. Encolado del tráfico

Para dirigir el tráfico a las bandas deseadas se usarán los filtros y clasificadores, que se llaman desde las qdisc empezando por la raíz (root). Una vez que se dirija el tráfico a una clase, se podrá volver a aplicar otro filtro correspondiente a su qdisc adjunta o simplemente dejar que los paquetes se encolen en la qdisc de la clase a la que los paquetes fueron dirigidos inicialmente.

4.3.2.2. Árboles: Relación entre qdiscs y clases

Más arriba mencionamos que los árboles están formados por qdiscs con clases y sin clases organizadas de forma jerárquica.

Para establecer una jerarquía hay que identificar a las qdiscs con una serie de controladores (handles). Se usarán para establecer cuáles serán las relaciones padre-hijo de las qdiscs y poder hacer referencia a ellas durante la etapa de filtrado y clasificación. Los controladores tienen un formato `<número_mayor>:<número_menor>`.

Todas las qdiscs tienen un par de parámetros usados para establecer la relación entre los elementos del árbol. Son “parent” y “handle”. Con el primero de ellos indicaremos cual es la clase superior (clase padre). Por otra parte, con “handle”, se establece el controlador que tendrá dicha qdisc. Para las clases en lugar de “handle” se usa “classid”.

Las clases deben tener el mismo número mayor que sus padres. Este número mayor tiene que ser único dentro de una configuración de salida o entrada. El número menor debe ser

único dentro de una qdisc y sus clases.
Un árbol de preferencias puede tener un aspecto parecido al que se expuso en el caso práctico.

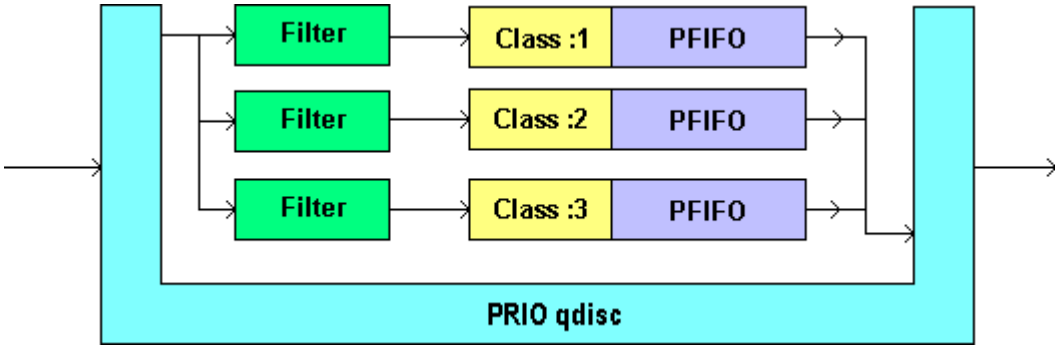
4.3.2.3. Desencolado del tráfico

Cuando hay que desencolar paquetes, el núcleo realiza la petición sobre la qdisc raíz de asociada a la interfaz. Hay que recorrer todo el árbol hasta encontrar la cola a desencolar.
Hay que entender que los paquetes a desencolar no viajan desde la qdisc (adjunta a la clase terminal correspondiente) hasta el driver de la interfaz de red. Lo que sucede en realidad es que se van escalando paquetes por el árbol mediante las relaciones que se establecieron previamente. Y aquí es dónde está la ventaja de las clases y el shaping que realizan la mayoría de ellas. Si una clase establece una determinada tasa, su clase hija o qdisc adjunta deberá cumplirla, por lo que se obligan a cumplir los ajustes establecidos en las clases padres.

4.3.2.4. PRIO

Esta qdisc en realidad no realiza ningún tipo de ajuste (shaping), sino que sirve para clasificar el tráfico usando sus clases.
Se puede decir que esta qdisc es como 'pfifo_fast' pero permitiendo ser manipulada con la herramienta TC y no limitándose sólo a la clasificación de paquetes por el campo TOS de los mismos.
Por defecto, al usar esta qdisc se crearán tres clases. La que tenga la numeración de prioridad más baja es la primera que tratará de desencolarse. Es decir, que si se quiere dar prioridad a cierto tráfico, habrá que enviarlo a las clases (colas) con más prioridad para desencolar.
Además, al crear inicialmente las clases, se asignarán a ellas unas qdisc puramente fifo.
Por otra parte, cabe destacar que en el caso de cambiar el número de clases que tendrá las qdisc, habría que modificar los valores del PRIOMAP. Lo que se consigue con ello es que en caso de no hacer filtrado y clasificación personalizados de los paquetes, se

puedan clasificar atendiendo al campo TOS del datagrama IP.



Funcionamiento de la Qdisc PRIO

Se puede obtener más información con '$man tc-prio'.

4.3.2.5. CBQ

Se trata de una qdisc bastante más compleja pero muy potente al mismo tiempo.

Permite hacer tanto shaping como scheduling. Es decir, podremos limitar el tráfico a la tasa que deseemos y además clasificarlo gracias a las clases que creemos con ayuda de los filtros.

No se explicará mucho más esta qdisc porque, aún siendo muy usada y clásica, no siempre se comporta del modo deseado debido a la forma en que trabaja. Además, resulta un poco difícil configurarla de forma correcta.

Esta qdisc funciona como una combinación de PRIO y TBF.

Para desencolar los paquetes se usa un WRR (Round Robin por pesos) o lo que es lo mismo, siempre que haya paquetes en las clases con más prioridad, éstos saldrán antes hacia la interfaz.

Se puede obtener más información sobre sus parámetros y funcionamiento en la página del manual: '$man tc-cbq'.

4.3.2.6. HTB (Hierarchical Token Bucket)

Esta qdisc con clases aparece para usarse en lugar CBQ. La razón de ello es dada la complejidad de esta última. Ambas realizan exactamente la misma función, pero HTB no realiza los cálculos de la misma manera que CBQ y resulta más estable.

HTB es como TBF pero además se le añaden clases. Es decir, se puede realizar shaping y scheduling.

A continuación se mencionan algunos de sus parámetros (los más relevantes):

1. default <número_menor>

   Es un parámetro opcional que sólo es posible utilizar cuando HTB se usa como qdisc y no como clase. Lo que indicamos con él es a qué clase (con 'classid' <número_menor>) deberán ir los paquetes que no hayan sido clasificados.

2. classid

   Al igual que las qdiscs pueden ser nombradas (identificadas) con el parámetro 'handle', las clases pueden hacerlo con 'classid'. Será útil del mismo modo que lo es 'handle' con las qdisc; para poder usar el identificador en la etapa de clasificación y filtrado.

3. prio

   Se trata del parámetro que indicará el peso de la clase en la que se utilice. Será útil para que el algoritmo de RoundRobin pueda desencolar el tráfico de forma “ordenada”. Podemos decir que aquí es donde se encuentra, casi en su totalidad, la fase de scheduling de HTB.

4. rate

   Es la tasa de transferencia que tiene garantizada la clase en la que se aplica, así como sus hijos (debido a cómo se desencolan los paquetes).

5. ceil

   Con este parámetro establecemos la máxima transferencia permitida la clase para la que se aplica. Si el padre tiene ancho de banda disponible, éste se le prestará a la clase llegando a un máximo establecido por ceil.

   Si se omite, se establece por defecto a la misma magnitud que 'rate'. Por lo que en

dicho caso no se prestará ancho de banda para esta clase.

Para obtener más ayuda e información, además de los parámetros restantes, se puede consultar tanto en la web del algoritmo HTB, como en el manual de debian:

http://luxik.cdi.cz/~devik/qos/htb/

$man tc-htb

4.3.3. La qdisc Ingress

Comentamos con anterioridad que sólo era posible controlar el tráfico que sale de una interfaz, aunque se podía “hacer algo” con el de entrada.

Se trata de aplicar una qdisc llamada 'ingress' a la interfaz deseada con la ventaja de que puede asignarse aún teniendo dicha interfaz alguna otra qdisc para el tráfico de entrada ('egress').

Lo que podemos hacer con esta qdisc es descartar los paquetes para mantener una tasa determinada. Se comporta igual que TBF, es porque los filtros de tc tienen la capacidad de trabajar de esta manera. Por lo tanto, podremos asignar valores de 'rate' y 'burst'.

4.4. Herramientas Iptables y TC

Usaremos este conjunto de herramientas para la clasificación y el filtrado del tráfico. Empezaremos por Iptables, que será con la herramienta que filtraremos el tráfico.

4.4.1. Iptables

No se explicará nada a parte de lo usado (mangle), por lo que si no se está muy familiarizado se puede empezar por:

http://www.pello.info/filez/firewall/iptables.html

Usaremos las tabla mangle, que es destinada a la manipulación de paquetes.
Nuestro objetivo es poner una especie de marca al paquete para luego poder dirigirlo a una determinada qdisc de nuestro QoS.

La tabla mangle de Iptables se usa como filter por ejemplo. Es decir, podemos indicar interfaces de entrada y salida, puertos origen y destino, diferenciar por TOS (Tipo de servicio), tipo de protocolo, etc.

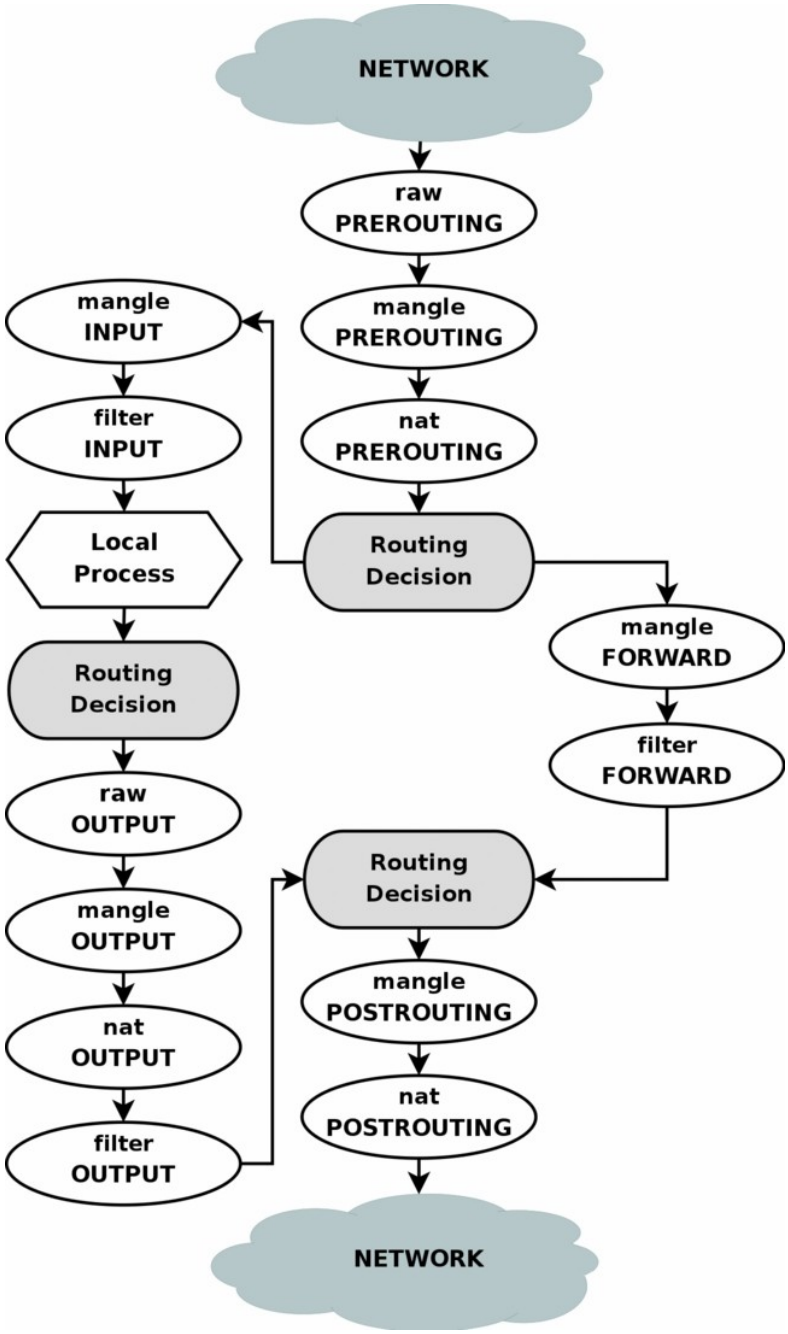
Para ver con facilidad cómo marcar un paquete pondremos un ejemplo. Marcaremos con 1 (podría ser 2, 3, 4, etc.) los paquetes que vayan dirigidos al puerto 80 y salgan por la interfaz eth0 de nuestro router:

```
iptables -t mangle -A POSTROUTING -o eth0 -p tcp --dport 80 -j MARK \
--set-mark 1
```

Una vez que apliquemos las marcas podemos comprobarlo y ver las veces que se han aplicado con el siguiente comando:

```
iptables -t mangle -vnL
```

En el siguiente diagrama se puede observar el orden en el que trabajan las tablas y cadenas por defecto de Iptables.



Según las necesidades se usa una cadena u otra. El esquema ayuda bastante para poder decidir.

4.4.2. TC

La herramienta TC (Traffic Control) se encuentra disponible a través del paquete *iproute* en Debian, por lo que está instalada con un sistema base.

Esta herramienta se puede usar para todos los pasos necesarios a la hora de implantar QoS, pero nosotros la descartaremos para la clasificación de paquetes, ya que lo realizaremos con Iptables.

Si no usásemos Iptables para clasificar paquetes podríamos usar, por ejemplo, el clasificador *U32* para buscar coincidencias en los paquetes.

Con TC añadiremos qdiscs a una interfaz, crearemos clases estableciendo parámetros y adjuntaremos qdiscs a esas clases. Además gestionaremos las marcas que realizamos con Iptables y las dirigiremos dependiendo de nuestras necesidades a una clase u otra.

A continuación se muestra la sintaxis de diferentes acciones del comando TC:

4.4.2.1. Qdiscs

```
tc qdisc [ add | change | replace | link ] dev IF [ parent id | root ] \
[ handle id ] qdisc [ parámetros_específicos ]
```

Ejemplo: Añadir una qdisc raiz a la interfaz HTB y redirigir el tráfico por defecto a 1: 40 .

```
tc qdisc add dev eth0 root handle 1:0 htb default 40
```

4.4.2.2. Clases

```
tc class [ add | change | replace ] dev IF parent id [ classid id ] \
qdisc [ parámetros_específicos ]
```

Ejemplo: Añadir una clase HTB a la qdisc raiz y limitar el tráfico a 600kbits .

```
tc qdisc add dev eth0 parent 1:0 classid 1:1 htb rate 600kbit
```

4.4.2.3. Filtros

```
tc  filter  [ add | change | replace ] dev IF [ parent id | root ] protocol \
protocolo prio prioridad tipoDeFiltro [ parámetros del filtro ] flowid id
```

Ejemplo: Filtrar un paquete que previamente fue marcado con un “1” mediante Iptables (ver ejemplo de Iptables) haciendo uso del clasificador “fw” diseñado para este tipo de combinación (TC+Iptables).

```
tc filter add dev $IF_EXT protocol ip prio 1 parent 1:0 handle 1 fw classid 1:1
```

Nota: 'Prio' en filter quiere decir en el orden en el que se aplicarán los filtros. Hasta que no se encuentra una coincidencia en los clasificadores éstos serán recorridos, como es lógico. Por lo tanto podemos cambiar la prioridad en los clasificadores acorde al tráfico que tengamos.

4.4.2.4. Mostrar qdiscs y clases aplicadas

```
tc [ FORMATO ] qdisc show [ dev IF ]
tc [ FORMATO ] class show [ dev IF ]
```

4.4.2.5. Eliminar qdiscs

Borrar la qdisc raíz (egress):

```
tc qdisc del dev $IF root
```

Borrar la qdisc ingress asociada a la interfaz:

```
tc qdisc del dev $IF ingress
```

5. Conclusiones

Al realizar este proyecto he visto la importancia que tiene implantar un QoS en lugares con altas tasas de tráfico en la red. Se ahorra dinero y se optimiza el rendimiento de la línea para las diferentes necesidades que se tengan.

Además de lo anterior y del uso de herramientas como *TC* e *Iptables*, he conseguido iniciarme en otros campos como son:

- Manejo básico de LaTeX.
- Creación de scripts “reales” de inicio en Debian.
- Creación de paquetes *.deb* .
- Monitorización de la red con MRTG.

Por mi parte creo que se han cumplido todos los objetivos y ha sido gratificante ver como el mecanismo que se ha implantado ha dado buenos resultados.

6. Posibles mejoras

Aún funcionando todo correctamente, opino que este proyecto puede mejorarse. Podrían plantearse los siguientes cambios:

- Uso de *ESFQ* en lugar *SFQ*.

  Usando esta Qdisc se darán las mismas posibilidades de salida a la red por IP y no por conexión. A día de hoy no está incluida en el Kérnel y ha sido por lo que no se ha usado. No es lógico tener que recompilar un núcleo de un sistema en producción.

- Uso del módulo *layer7* de Iptables.

  Ocurre lo mismo que con ESFQ. Es un módulo que todavía no está incluído ni en el Kérnel ni en Iptables. Por lo tanto, habría que recompilar tanto el Kérnel como Iptables para poder usarlo.

  Por otra parte, habría que disponer de una máquina potente, porque *layer7* desencapsula los paquetes hasta el nivel de aplicación. Es muy interesante pero resulta costoso computacionalmente hablando.

- Uso de una aplicación de monitorización más avanzada.

  Una vez que se implantó MRTG se observó que está diseñado para conexiones más simétricas, por lo que para nuestro caso no es del todo útil en cuanto a la observación de la subida.

  Existen soluciones como *PandoraFMS*, *Cacti*, *Nagios*, etc.

- Ajuste más fino del QoS.

  A la hora de implantar el QoS, no ha habido mucha actividad en la red de los ciclos, por lo que no se sabe cómo responderá la red exactamente cuando haya mucha carga.

  Habrá que ir observando el funcionamiento y mejorarlo poco a poco.

7. Bibliografía

- Linux Advanced Routing & Traffic Control.
  http://lartc.org

- Manual en PDF de un usuario.
  http://usuarios.multimania.es/ccd_illusions/QoS-3.pdf

- Otro manual de TC.
  http://pupa.da.ru/tc/

- Imágenes de Qdiscs.
  http://www.kiv.zcu.cz/~simekm/vyuka/pd/zapocty-2004/traffic-dvorak/

- Manual de HTB.
  http://luxik.cdi.cz/~devik/qos/htb/manual/userg.htm

- Wiki de Debian.
  http://wiki.debian.org/

- Documentación de MRTG.
  http://oss.oetiker.ch/mrtg/doc/index.en.html

- Páginas del *man*.