Departamento de Tecnología Electrónica
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Laboratorio 5 - QoS y Control de tráfico

Enunciados de laboratorios
Tecnologías Avanzadas de la Información
Rev. 5 - 21/12/21

1. Introducción y objetivos

El tiempo estimado de realización de este laboratorio es de **4 horas** y está centrado en presentar sistemas de control de tráfico relacionados con QoS en Linux. Linux implementa en su núcleo mecanismos para gestionar el tráfico con multitud de posibilidades. Algunas de las implementaciones incluidas no están ampliamente probadas, y de otras se conocen algunas deficiencias en su funcionamiento. A pesar de ello, conociendo las limitaciones existentes se pueden conseguir resultados satisfactorios tras realizar pruebas con diferentes configuraciones. Ante la multitud de posibilidades existentes, en este laboratorio se describirán sólo algunas probando un conjunto reducido de ellas, concretamente las más relacionadas con la parte teórica de la asignatura.

La parte práctica de este laboratorio se ha dividido en dos grandes bloques, el primero está completamente guiado para facilitar la comprensión de la configuración y de los resultados obtenidos al aplicar diferentes disciplinas. En un segundo bloque, se requiere realizar la configuración de una disciplina descrita esquemáticamente donde se deben aplicar los conocimientos adquiridos. Como en el resto de los laboratorios, para facilitar el desarrollo de la sesión se debe disponer del material de la tabla 1 en la ubicación indicada.

	Descripción	**Ubicación**
descargas.sh, descargas2.sh	Script para realizar transferencias simultáneas	Equipo local o Máquina compartida
escucha-puerto.sh	Script para emular un servicio en un puerto TCP	GW

Tabla 1. Material necesario para la realización de la sesión de laboratorio.

2. Implementación en Linux del control de tráfico

La documentación de referencia para técnicas avanzadas de red en Linux es LARTC (Linux Advanced Routing & Traffic Control) disponible en https://www.lartc.org/. Esta documentación es extensa, algunas partes están incompletas y se recomienda utilizar la versión en inglés, ya que las traducciones no son de buena calidad. La mayor parte del contenido aquí expuesto se encuentra a lo largo del capítulo 9 de esta guía, pero se añaden en esta documentación algunas consideraciones fruto de la experiencia con el uso de estas herramientas de Linux que no se tratan en la documentación oficial.

La implementación del control de tráfico en Linux presenta cierta complejidad, pues está pensada para facilitar a los desarrolladores la implementación de nuevas disciplinas para el control del tráfico. De hecho, aparecen en el núcleo multitud de tipos de colas y disciplinas con funcionalidad diferente y desarrolladas por diferentes personas, e incluso, algunas experimentales no ampliamente probadas. A pesar de la diversidad, todas las implementaciones son homogéneas respecto al modo de configuración, realizándose todas con la misma herramienta (*tc*) facilitando al administrador del sistema su configuración. Esta herramienta está ampliamente documentada en las páginas de manual de Linux.

Básicamente, el control de tráfico en Linux realiza cuatro operaciones: conformado de tráfico, reordenación de paquetes, vigilancia y eliminación de paquetes. Estas operaciones se realizan configurando cada interfaz de red con múltiples disciplinas, cada disciplina puede llevar asociada una o varias colas, y los paquetes que llegan a la interfaz se ubican en alguna de las colas existentes. La cola inicial para cada paquete depende de la clasificación. El núcleo decide cual es la disciplina y la cola inicial para cada paquete mediante listas de reglas, llamados filtros.

En la implementación realizada en Linux las disciplinas de colas que se deseen usar se asocian en forma de árbol, formando una jerarquía de disciplinas cuyo nodo raíz es una interfaz de red. Posteriormente a ciertos nodos se adjuntan reglas para decidir cual es el nodo destino de cada paquete, estas reglas se denominan filtros. Algunas disciplinas de colas van más allá de ser un simple nodo de la jerarquía, incluyen subnodos llamadas clases hijas donde se pueden encolar los paquetes.

En la figura 1 se muestra esquemáticamente una posible configuración con diferentes disciplinas y clases. La clave para entender este mecanismo es profundizar en los tres tipos de elementos que forman parte del árbol: disciplinas, clases y filtros.

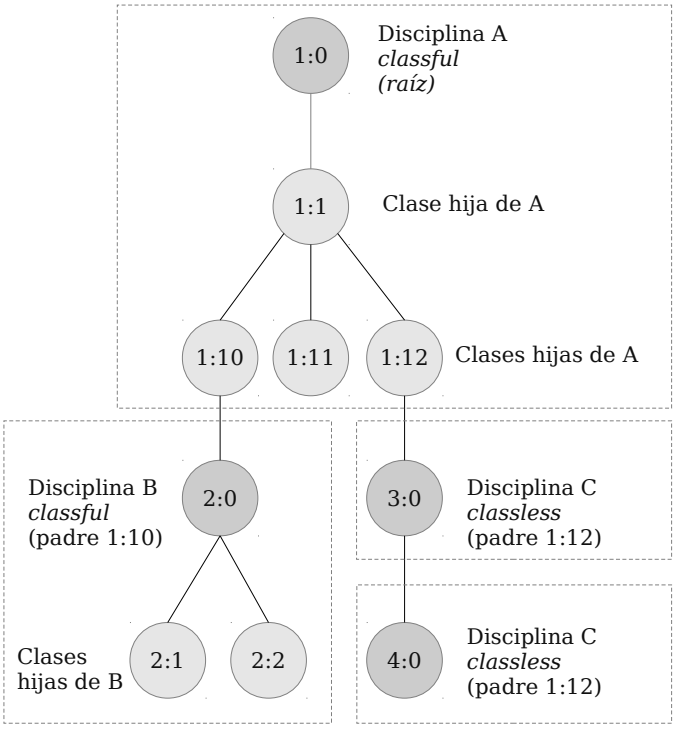

Figura 1. Jerarquía de ejemplo

Las clases siempre forman parte de una disciplina, no pueden existir fuera de ellas. Es fácil de entender con el siguiente ejemplo: suponga que se desea utilizar una *disciplina de colas de prioridad* con 5 colas de prioridad diferentes; en vez de crear 5 disciplinas en el árbol, la disciplina *cola de prioridad* incluye la posibilidad de añadir clases hijas donde, cada clase hija contiene una cola diferente con una prioridad asignada. Así, con sólo una disciplina se consigue esta configuración.

Pero no todas las disciplinas implementadas disponen de clases hijas, distinguiéndose dos tipos: disciplinas sin clasificación (*classless*) y disciplinas con clasificación (*classful*). Para no crear confusión entre *classless* y *classful* se define una disciplina *classless* como aquella que no puede subdividirse en clases hijas. Para las disciplinas que contienen clases, estas clases aparecen en el árbol de jerarquía como nodos hijos de las disciplinas quedando el árbol formado sólo por dos tipos de nodos, las disciplinas y las cases hijas de las disciplinas. Es importante resaltar que todas las disciplinas de colas en tienen funcionalidades simples y son: aceptar los datos, reordenar su envío, retrasarlo o eliminarlo de la cola. Como ya se indicó, toda esta funcionalidad se manipula con el comando *tc*, y por cada disciplina disponible existe una página de manual adicional donde se precisa la configuración de la misma y de las posibles clases hijas.

Respecto a la clasificación de los paquetes, anteriormente se han presentado los filtros como reglas de clasificación. Para utilizar correctamente los filtros se debe conocer como el núcleo de Linux interactúa con el árbol jerárquico de disciplinas creado por el administrador del sistema. Siempre, el punto de entrada de los paquetes es la raíz de la jerarquía, así, cuando los paquetes llegan a una interfaz se les aplican los filtros asociados a la raíz de la jerarquía. Con estos filtros se decide en que clase de todo el árbol se encola el paquete. Aunque no se recomienda, también es posible tener filtros asociados a otros nodos internos de la jerarquía y con este tipo de configuración, cuando un nodo inferior recibe un paquete pueden de nuevo aplicarle más filtros asociados al nodo siendo de nuevo movido a otro hijo dentro de la jerarquía. Esta solución compleja sólo tiene sentido para su aplicación en situaciones donde existen multitud de filtros. El sistema de filtrado se optimiza distribuyendo filtros por el árbol ya que disminuye el tamaño de las listas de filtros en cada nodo.

Otro aspecto importante es conocer como el sistema de filtrado identifica los nodos inequívocamente para poder ubicar cada paquete en un nodo. La solución adoptada consiste en usar un identificador único en cada elemento de la jerarquía, este identificador único es una pareja de números llamados número mayor y número menor, separados por “:”. Éstos serán asignados explícita o implícitamente durante la configuración. Los números mayores siempre identifican las disciplinas, así, cada disciplina tendrá un número mayor diferente. En cambio, el número menor identifica a cada clase hija de una disciplina, teniendo siempre cada clase hija el número mayor de la disciplina a la que pertenece.

Una vez vistos los aspectos generales las disciplinas de colas, se enumeran las disciplinas sin clasificación (*classless*) más relevantes disponibles:

- PFIFO y BFIFO: Disciplina simple basada en una cola FIFO de un tamaño fijo en paquetes (PFIFO) o bytes (BFIFO).
- PFIFO_FAST: Disciplina estándar consistente en una cola FIFO dividida en tres bandas de

prioridad relacionadas con el campo TOS de los paquetes IPv4.

- RED: *Random Early Detection*, disciplina para simular la congestión de un enlace eliminando paquetes a azar.
- SFQ: *Stochastic Fairness Queueing*, es una disciplina llamada estocástica equitativa basada en la auto-detección de flujos, reordena los paquetes para tratar equitativamente a cada flujo.
- TBF: *Token Bucket Filter*, disciplina de cubeta con fichas para regulación de velocidad.
- CHOKE: *CHOose and Keep for responsive flows*, disciplina para usar en situaciones de congestión, identifica y penaliza los flujos que monopolizan el enlace.

Las disciplinas de colas con clasificación (*classful*) más relevantes son:

- PRIO: Disciplina de cola con bandas de prioridad configurables, no implementa regulación de velocidad.
- CBQ: *Class Based Queueing*, es una disciplina con clases hijas para compartir el ancho de banda, compleja y con multitud de parámetros como son: prioridad, límites de ancho de banda, etc.
- HTB: *Hierarchy Token Bucket*, esta disciplina también sirve para compartir el ancho de banda entre clases hijas garantizando ancho de banda en cada clase además de priorizar y regular velocidad basado en TBF.
- DRR: *Deficit Round Robin Scheduler*, es una disciplina con mayor flexibilidad pensada para reemplazar a SFQ, la diferencia es que DDR no contiene de manera predeterminada ninguna cola. Básicamente su funcionamiento consiste en descartar cualquier paquete que no sea clasificado por un filtro.

Además de las enumeradas anteriormente existen algunas otras que se pueden consultar en la documentación de Linux y en la página de manual del comando *tc*.

En las siguientes secciones se mostrarán los aspectos más relevantes de las diferentes disciplinas pero entrando sólo a detallar aquellas utilizadas en la sección práctica. Para poder entender los ejemplos de configuración, en primer lugar se describe brevemente la herramienta *tc* que se debe utilizar como administrador para la configuración.

2.1. Herramienta TC

Toda la configuración del control de tráfico se realiza con el comando *tc*. El uso general de este comando tiene tres modos y cada uno permite manipular un elemento diferente del sistema de control de tráfico: disciplinas, clases y filtros.

La sintaxis para configurar disciplinas y clases depende de cada disciplina usada, y se mostrarán diversos ejemplos en la descripción de cada disciplina considerada de interés. No obstante, los filtros funcionan de manera similar para todas las disciplinas y se dedicará una sección adicional en este documento para explicarlos debido a su complejidad.

De forma general, se utilizará el comando *tc* según lo indicado en la tabla 2, y se debe considerar un aspecto adicional que son las unidades utilizadas con este comando, ya que pueden crear confusión. En la tabla 3 se muestra el significado de las unidades, las cuales, se puede consultar en la página principal de manual de *tc*.

Modo	Comando (ejemplo)	Descripción
Disciplina	**tc qdisc add dev** eth0 **root** tbf ...	Añadir disciplina TBF como raíz
	tc qdisc add dev eth0 **parent** 1:0 **handle** 10: sqf ...	Añadir disciplina SFQ como hija de la disciplina 1:0
	tc qdisc del dev eth0 **root**	Eliminar todas las disciplinas de la interfaz eth0 (limpieza)
Clase	**tc class add dev** eth0 **parent** 1:0 **classid** 1:1 **htb** ...	Añadir clase hija a la disciplina 1:0
Filtro	**tc filter add dev** eth0 **parent** 1:0 **protocol** ip ...	Añade un filtro para clasificación a la disciplina 1:0
Estadísticas	**tc** -s -d **qdisc show dev** eth0	Muestras las disciplinas de la interfaz eth0
	tc -s -d **class show dev** eth0	Muestra las clases de la interfaz eth0
	tc -s -d **filter show dev** eth0	Lista los filtros de la interfaz eth0

Tabla 2. Modos de operación del comando tc.

Sintaxis	Unidad	Equivalencia
bps	Bytes por segundo / Bytes	
kbps	Kilobytes por segundo / Kilobytes	1000bps
mbps	Megabytes por segundo / Megabytes	10^6bps
gbps	Gigabytes por segundo / Gigabytes	10^9bps
bit	Bits por segundo	
kbit	Kilobits por segundo / Kilobits	1000bits
gbit	Gigabits por segundo / Gigabits	10^6bits
mb	Megabytes	10^9bits
s, sec, secs	Segundos	
ms, msec, msecs	Milisegundos	10^{-3} sec
us, usec, usecs	Microsegundos	10^{-6} sec

Tabla 3. Unidades utilizadas en el comando tc.

2.2. Disciplinas sin clasificación

Las disciplinas sin clasificación no disponen de la posibilidad añadir clases hijas para clasificar los paquetes con filtros. Estas disciplinas tienen dos usos generales: en la raíz de la interfaz y en las hojas

del árbol de clasificación.

Estas disciplinas se utilizan conjuntamente con disciplinas que sí admiten clasificación, de forma que una disciplina con clasificación y múltiples clases hijas admite una disciplina sin clasificación debajo de sus clases de último nivel. Se mostrará la utilidad de este tipo de configuración en la sección práctica.

2.2.1. Disciplina PFIFO / BFIFO

Las disciplinas más básicas existentes son PFIFO y BFIFO y son simples colas FIFO cuya política de envío es *Fisrt In – Fisrt Out*. La primera letra P o B hace referencia a paquetes o bytes respectivamente. En su configuración sólo admite como parámetro el tamaño de la cola (parámetro *limit*), y a pesar de su simpleza, su uso es recomendado en múltiples situaciones. Otra característica importante es que mantiene estadísticas del funcionamiento de la cola y pueden consultarse fácilmente. Una disciplina de este tipo puede ser añadida mediante el comando:

```
tc qdisc add dev eth0 root pfifo limit 10
```

2.2.2. Disciplina PFIFO_FAST

Es la disciplina utilizada en Linux de forma predeterminada en cada interfaz, Como en el caso anterior, la política de envío es *Fisrt In – Fisrt Out* pero, con una modificación consistente en la división en 3 partes de la cola, llamadas bandas. No se estudiará en profundidad ya que no se usará en esta sesión de laboratorio, pero se resumen algunos detalles sobre su modo de operación y se puede consultar una amplia documentación en la página de manual *tc-pfifo_fast*.

El funcionamiento básico de esta disciplina es la siguiente: el núcleo minimiza el retraso para los paquetes de la banda 0, por ello, mientras hay paquetes en la banda 0, las bandas 1 y 2 no son procesadas, y para procesar la banda 2 deben estar vacías las bandas 0 y 1. Los paquetes son introducidos automáticamente en las bandas según una configuración establecida por el administrador (parámetro *priomap*) consistente en mapear cada uno de los posibles valores del campo TOS del paquete a cada banda, al existir 4 bits TOS aparecen 16 valores para mapear.

Esta disciplina se confunde a veces con una disciplina *classful*, pero no lo es, la diferencia está en que una disciplina *classful* puede contener tantos filtros añadidos como se desee para analizar los paquetes. En cambio PFIFO_FAST no admite filtros, se asignan paquetes a cada banda analizando únicamente el campo TOS, no se puede establecer otro tipo de filtro para realizar una clasificación adicional.

En la documentación de esta disciplina se detalla cual es el mapa de prioridad establecido de manera predeterminada. Este mapa está relacionado con el significado de los cuatro bits TOS: minimizar retraso, maximizar caudal, maximizar fiabilidad y minimizar coste.

2.2.3. Disciplina cubeta con fichas TBF

Esta disciplina ha sido ampliamente estudiada en la parte teórica de la asignatura y es implementada en Linux con algunas peculiaridades. La disciplina TBF (*Token Bucket Filter*) controla el caudal de salida con gran precisión y permite pequeñas ráfagas bajo ciertas condiciones.

Para hacer un uso adecuado de esta disciplina se debe establecer correctamente la velocidad de entrada de fichas y el tamaño de la cubeta, siendo consciente del efecto que tienen estos parámetros:

- La velocidad de entrada de fichas regula con precisión la velocidad de salida.
- El tamaño de la cubeta indica la cantidad máxima de paquetes que pueden salir en una ráfaga.

Con la implementación realizada en Linux se contempla una correspondencia entre fichas y bytes en relación uno a uno, es decir, una ficha es un byte. Además de los parámetros básicos como tamaño de cubeta, tamaño de cola y fichas por segundo, esta implementación contempla parámetros adicionales con el objetivo de aumentar la precisión sobre el control del flujo de salida de la cubeta. Es fundamental utilizarlos adecuadamente y comprender el efecto causado en el flujo de salida la alteración de cada uno de ellos. Son los siguientes:

- Tamaño de cola o latencia (*limit / latency*): establece el tamaño de la cola, es posible en la configuración indicar la latencia en vez del tamaño. Dada una latencia, el sistema calcula el tamaño de cola automáticamente.
- Tamaño máximo de ráfaga (*burst*): corresponde al tamaño de la cubeta en bytes. Si se establece este parámetro muy pequeño se perderán muchos paquetes que no cabrán en la cola si entra una ráfaga, y no se regulará la velocidad correctamente. Según la documentación se debe considerar una relación mínima en enlaces de 10Mbits/s al menos 10KBytes.
- Tamaño mínimo del token (*mpu*): este parámetro se utiliza para considerar que los paquetes sin datos sí consumen ancho de banda, por ejemplo, el tamaño mínimo de la trama ethernet, incluyendo únicamente cabeceras, es de 72 bytes. Debe establecerse al valor en bytes consumido por un paquete vacío.
- Velocidad o tasa (*rate*): Número de bytes o bits (según unidad usada) por segundo con el que se llena la cubeta.
- Velocidad pico (*peakrate*): Este parámetro tiene sentido cuando la cubeta contiene fichas y se emite una ráfaga. Se podrían quitar todas las fichas y los paquetes en la mínima unidad de tiempo de computación y se obtendría una velocidad de ráfaga prácticamente infinita, produciendo efectos indeseados. Este parámetro habitualmente no es necesario establecerlo al ser automáticamente calculado a partir de los otros.
- Cubeta de ráfaga (*mtu/minburst*): El parámetro anterior (*peakrate*) limita la velocidad de la ráfaga, así surge una cubeta secundaria encargada de limitar la velocidad de ráfaga. Se debe establecer al tamaño MTU de la interfaz de red para un comportamiento óptimo.

En la mayoría de los casos no es necesario afinar la configuración estableciendo todos los parámetros descritos. De hecho, muchos de ellos son calculados automáticamente, así que para una configuración

típica en una disciplina TBF, basta con especificar los indicados en el siguiente ejemplo:

```
tc qdisc add dev eth0 root tbf rate 630kbit latency 50ms burst 1540
```

2.2.4. Cola estocástica equitativa SFQ

Este tipo de disciplina es una implementación de la disciplina CFQ estudiada en la parte teórica de la asignatura, presenta algunas diferencias en la implementación. Con CFQ se establecían diferentes colas de envío equitativas, donde un sistema de clasificación se encarga de añadir los paquetes en cada cola. Usando un mecanismo *round robin* con un *quantum* determinado se extraían paquetes de cada cola de forma circular. La diferencia es que con SFQ las colas son creadas automáticamente y el tráfico también es clasificado automáticamente. Esta disciplina es muy recomendable en enlaces de salida congestionados ya que equilibra todos los flujos.

La implementación realizada en Linux realiza una auto-clasificación de flujos, el comportamiento es equivalente a crear de forma dinámica una cola diferente para cada cada flujo (sesión) existente, ya sea TCP o UDP, y a las colas se les aplica el algoritmo CFQ. Con esta implementación se obtiene un tratamiento equitativo para cada flujo. En la implementación se simplifican la colas, no creando en realidad una cola para cada flujo, se utilizan funciones *hash* para identificar y clasificar paquetes de cada flujo, todos los paquetes residen en una estructura de datos optimizada para su acceso con las funciones *hash*.

El uso de funciones *hash* para selección de paquetes rompe la equidad en intervalos de tiempo grandes, por ello, esta implementación cambia cada cierto tiempo la función *hash* utilizada, este tiempo de cambio de función es considerado un parámetro configurable en este tipo de disciplina.

La configuración de una disciplina SFQ es simple y sólo hay que contemplar estos parámetros:

- Tiempo de cambio de la función hash (*perturb*): De forma predeterminada se establece a 10 segundos, no se recomienda cambiarla aunque los valores en el intervalo 5-10 son razonables.
- Quantum: Cantidad de bytes extraídos de cada flujo en el turno, es importante no establecer este parámetro por debajo de la MTU, podría quedarse un flujo sin recibir servicio.
- Tamaño de cola (*limit*): Cantidad de paquetes totales que pueden almacenarse en la cola.
- Divisor: Permite establecer el tamaño de tabla *hash*, no se recomienda su utilización.

Como en casos anteriores se pueden omitir parámetros en la configuración y dejar que el sistema los establezca adecuadamente, un ejemplo sería el siguiente:

```
tc qdisc add dev eth0 root sfq perturb 10
```

2.3. Disciplinas de colas con clasificación

Estas disciplinas son más complejas y admiten filtros para clasificar los paquetes a lo largo del árbol de clases hijas. A su vez, las clases hijas admiten como hijos otras disciplinas adicionales siendo esta configuración, aunque parezca compleja, de uso muy común ya que aporta beneficios en el comportamiento de los flujos.

Se puede generalizar el comportamiento de cada clase como un conformador y/o planificador de tráfico: regula la velocidad de salida y establece un orden de salida de paquetes. Existe una disciplina llamada PRIO cuya finalidad no es la regulación, sólo la reordenación en la salida de paquetes y otras como CBQ y HTB que implementan tanto la regulación como la planificación.

Las principales disciplinas clasificadoras de este tipo son las tres mencionadas: PRIO, CBQ y HTB pero se estudiará en mayor profundidad HTB, por un lado porque presenta mejoras significativas respecto a CQB, tanto a nivel de rendimiento como a nivel de configuración, y por otro lado, porque existe configuraciones de HTB equivalentes a PRIO.

2.3.1. Disciplina PRIO

El comportamiento de esta disciplina es similar a PFIFO_FAST pero se pueden especificar el número de bandas deseadas. Por cada banda, automáticamente se crea una clase donde poder clasificar los paquetes. Cuando se configura la disciplina se puede indicar el número de bandas y el mapa de prioridad, aunque si no se especifican las bandas, se establecen tres y un mapa predeterminado. Así, los parámetros de configuración son *bands* y *priomap* respectivamente.

En el siguiente ejemplo se crean 4 bandas de prioridad estableciendo como número mayor de esta disciplina el número 1:

```
tc qdisc add dev eth0 root handle 1: prio bands 4
```

La ejecución de este comando crea la estructura mostrada en la figura 2 con 4 bandas de prioridad. Los números menores de las clases hijas son asignados automáticamente desde 1 a 4. Recuerde que el comportamiento es como el de PFIFO_FAST, las bandas de menor número son las más prioritarias y el modo de operación consiste en atender los paquetes de cualquier banda, si y sólo si, no hay paquetes en las bandas más prioritarias (aquellas con inferior número menor).

Se puede consultar la configuración establecida de clases por el sistema solicitando información con la herramienta *tc* sobre las disciplinas y clases existentes. Considere el ejemplo de la salida mostrada por los dos siguientes comandos y la correspondencia con la figura 2:

```
# tc -d qdisc show dev eth0
qdisc prio 1: root refcnt 2 bands 5 priomap  1 2 2 2 1 2 0 0 1 1
```

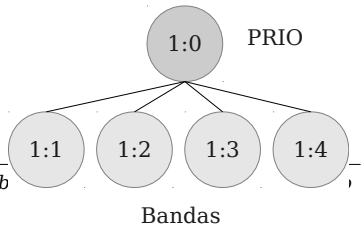

Figura 2. Disciplina PRIO con 4 bandas

```
1 1 1 1 1 1

# tc -d class show dev eth0
class prio 1:1 parent 1:
class prio 1:2 parent 1:
class prio 1:3 parent 1:
class prio 1:4 parent 1:
class prio 1:5 parent 1:
```

2.3.2. Disciplina CBQ

CBQ fue uno de los primeros clasificadores implementados en Linux pero ampliamente criticado tanto por desarrolladores como administradores de sistemas. Desde el punto de vista del administrador su utilización y configuración es extremadamente compleja, y desde el punto de vista de implementación se comprueba que no aprovecha en su totalidad el ancho de banda disponible y sobrecarga en exceso el sistema.

Básicamente CBQ consigue la regulación de velocidad dejando de emitir paquetes durante ciertos intervalos de tiempo, se dice que deja en reposo su salida. Las configuraciones realizadas con CBQ se pueden realizar de manera equivalente con HTB. HTB presenta dos ventajas principales frente a CBQ, la primera es una mayor facilidad de configuración, y la segunda, es una mayor eficiencia en el uso de CPU al realizar los cálculos de manera simplificada respecto a CBQ.

Por ello, no se detallará su uso y se centrará el laboratorio en el uso de HTB.

2.3.3. Disciplina HTB

Esta disciplina es óptima cuando se dispone de un ancho de banda fijo y estable, configurando HTB adecuadamente se puede dividir el caudal disponible en partes con una gran exactitud. El modo de operación de HTB consiste en garantizar un ancho de banda mínimo configurado para cada una de las subdivisiones, si hay excedente de ancho de banda, se cede a otras clases el ancho de banda sobrante pero en el reparto del sobrante, sólo se permite a cada clase alcanzar un máximo establecido en la configuración. Además de controlar el máximo/mínimo caudal en cada clase se implementa un sistema de prioridades en el reparto de ancho de banda sobrante consiguiéndose comportamientos equivalentes a la disciplina PRIO.

Básicamente el funcionamiento se puede presentar como una jerarquía de clases, donde las clases padres reparten ancho de banda a las clases hijas, siguiendo determinadas reglas. Para configurar esta disciplina se parte de un nodo raíz HTB que admite una única configuración que es la clase predeterminada donde se encola el tráfico. Tras esto, se configuran todas las clases hijas que se deseen en forma de árbol y en cada uno de los hijos del árbol se establecen los siguientes parámetros: caudal mínimo garantizado, caudal máximo alcanzable y prioridad en la adquisición de caudal disponible.

Para hacer un uso adecuado de HTB se requiere conocer el algoritmo seguido cuando se extrae un

paquete para su envío desde alguna clase del árbol HTB. El primer principio de operación es conocer una de las restricciones de HTB: todos los paquetes residen siempre en colas ubicadas en las hojas del árbol de clases HTB, por tanto, en los nodos internos nunca existen colas de paquetes. Una hoja es un nodo del árbol HTB que no tiene hijos.

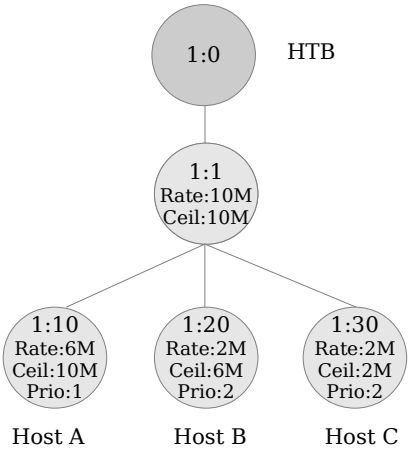

Figura 3. Jerarquía de ejemplo HTB

Con la restricción indicada, el algoritmo de operación en primer lugar reparte el caudal disponible en la jerarquía HTB siempre de padres a hijos y de dos formas: verticalmente entre clases padres e hijas y horizontalmente entre clases hermanas. Las reglas de distribución de caudal seguidas son las siguientes:

- Una clase hija siempre envía paquetes con el caudal mínimo garantizado.
- Una clase hija solicita ancho de banda al nodo padre para intentar llegar a su máximo caudal.
- Un padre suma los caudales que está recibiendo de sus hijos y si es menor que su caudal máximo reparte el sobrante entre los hijos.

En segundo lugar, el algoritmo usa un método de reparto del ancho de banda sobrante desde el padre a los hijos siguiendo un proceso basado en prioridades. Así, cada clase hija tiene configurada una prioridad, y en igualdad de prioridad se aplica *round robin* entre hijos con un *quantum* precalculado. Aunque el *quantum* puede ser configurado manualmente, no se recomienda.

Tras estudiar el modo de operación, el proceso de configuración también presenta algunas peculiaridades. Para realizarlo correctamente se indican las principales restricciones de una jerarquía HTB en la estructura del árbol de clases:

- Los nodos intermedios del árbol no pueden contener colas de paquetes, por ello, el destino de los filtros no pueden ser estos nodos.
- En las hojas del árbol (nodos que no tienen hijos) se puede indicar la disciplina de cola a usar, aunque de forma predeterminada se establece una cola PFIFO.

Para entender mejor lo expuesto se analiza la configuración HTB mostrada en la figura 3. En este ejemplo se distribuye un ancho de banda de 10MB entre tres equipos A, B y C a los que se aplican diferentes políticas. Los parámetros utilizados por HTB son *rate, ceil* y *prio* cuyo significado es:

- Rate: Caudal garantizado.
- Ceil: Caudal máximo alcanzable.
- Prio: Prioridad en la obtención del caudal sobrante.

Con la configuración y parámetros mostrados en la figura 3 cada equipo tiene la siguiente política asignada:

- Host A:
  - Tiene garantizado un caudal de 6MB.
  - Puede llegar a utilizar un máximo caudal de 10MB si hay caudal disponible.
  - Este equipo tiene prioridad 1 en la solicitud de ancho de banda sobrante a la clase padre.
- Host B:
  - Tiene garantizado un caudal de 2MB.
  - Utilizará un máximo caudal de 6MB si hay caudal disponible en el nodo padre.
  - Este equipo tiene prioridad 2 en la solicitud de ancho de banda sobrante.
- Host C:
  - Tiene garantizado un caudal de 2MB.
  - Utilizará un máximo caudal de 2MB.
  - Nunca solicita caudal extra a la clase padre por tener su caudal máximo igual al garantizado.

Fíjese en la restricción establecida en la configuración del equipo C, tiene el mismo valor establecido para el caudal garantizado que para el caudal máximo, esto significa que no participará nunca en el proceso de solicitud de ancho de banda a la clase padre ya que nunca pedirá caudal extra a la clase padre.

Para ilustrar con mayor profundidad el ejemplo de la figura 3, a continuación se describe el comportamiento de esta configuración en algunas situaciones, considere que existen más posibilidades de comportamiento para los tres equipos a parte de los siguientes tres casos:

- Caso 1: Todos los equipos intentan utilizar el máximo ancho de banda: como *(Rate 1:10) + (Rate 1:20) + (Rate 1:20) = (Rate 1:1)* la clase 1:1 no dispone de ancho de banda adicional para distribuir entre los hijos. El resultado es que todos los equipos disponen sólo del ancho de banda garantizado.
- Caso 2: El equipo C está en reposo, A y B intentan utilizar el máximo ancho de banda posible: como *(Rate 1:10) + (Rate 1:20) = 8MB* la padre clase 1:1 dispone de 2MB adicionales para repartir entre los hijos. La clase 1:1 reparte el caudal sobrante entre los hijos, pero por orden de prioridad, hasta que cada hijo alcance a su parámetro *Ceil*, por tanto, cede los 2MB a clase 1:10. El resultado es que el equipo A usa un caudal de 8MB y el equipo B sólo dispone de su caudal garantizado de 2MB.
- Caso 3: El equipo A está en reposo, C y B intentan disponer del máximo caudal posible. En esta situación *(Rate B) + (Rate C) = 4M*, así, la clase padre 1:1 dispone de 6MB adicionales para repartir entre hijos. El hijo más prioritario (A) no solicita caudal y los dos restantes, con igual prioridad solicitan hasta alcanzar su parámetro *Ceil*. El resultado en esta situación es que el equipo B usa un caudal de 6MB y el equipo C 2MB, ambos alcanzan el caudal *Ceil* (máximo) y a la clase padre le sobran 2MB que no utiliza.

Existen algunas situaciones en las que el comportamiento puede ser el no deseado, en el último caso sobra caudal debido a las restricciones impuestas a los equipos. Suponga ahora que la suma de los

caudales garantizados (*rate*) de todos los hijos es mayor que el caudal garantizado del padre; esta situación es no deseable y debe evitarse, ya que pierde sentido el parámetro *rate* como caudal garantizado, al no poder la clase padre garantizar el caudal de los hijos.Para completar el ejemplo se muestra la secuencia de comandos *tc* para conseguir la configuración mostrada en la figura 3:

```
tc qdisc add dev eth0 root handle 1: htb default 30
tc class add dev eth0 parent 1: classid 1:1 htb rate 10mbit
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 6mbit ceil 10mbit prio 1
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 2mbit ceil  6mbit prio 2
tc class add dev eth0 parent 1:1 classid 1:30 htb rate 2mbit ceil  2mbit prio 2
```

La existencia de la clase 1:1 es otra de las restricciones de HTB, se debe a que sólo se puede conformar tráfico en las clases hijas. En la especificación de la disciplina HTB sólo se puede indicar la clase por defecto pero no ningún otro parámetro relacionado con el conformado del tráfico. Con esto se concluye que siempre la disciplina HTB está obligada a tener al menos una clase hija donde se establezcan los parámetros de configuración *rate, ceil*, etc.

Usando adecuadamente esta disciplina existen equivalencias a las disciplinas TBF y PRIO presentadas con anterioridad, las equivalencias son las siguientes:

- Conseguir una disciplina TBF con HTB es equivalente a tener una clase HTB con el parámetro *rate* y *ceil* de igual valor, este ejemplo se mostró en el ejemplo de la figura 3.
- Conseguir una disciplina PRIO con HTB es equivalente a crear una clase hija HTB por cada banda de prioridad estableciendo los parámetros *prio* de cada clase hija adecuadamente.

Para finalizar se muestran algunos parámetros adicionales permitidos en la configuración de las clases HTB. Cuando se omiten parámetros durante la configuración, el sistema los establece a valores precalculados, pero en algunas situaciones es útil alterar su valor. Los parámetros indicados en la tabla 4 son los principales para esta disciplina.

Parámetro	Descripción
parent [mayor:menor]	Nodo padre en la jerarquía
classid [mayor:menor]	Identificador único del nodo en la jerarquía
prio [prioridad]	Mapa de prioridades para el proceso *round robin* de reparto entre las clases hijas
rate [caudal]	Caudal garantizado para esta clase (y sus hijos)
ceil [caudal]	Caudal máximo alcanzable de envío para esta clase (y sus hijos)
burst [bytes]	Tamaño en bytes de la ráfaga que puede superar el caudal máximo
cburst [bytes]	Tamaño en bytes de la ráfaga que se puede emitir instantáneamente, es decir directo a la interfaz a máxima velocidad
quantum	Cantidad de bytes usados en el reparto *round robin* entre clases hijas de misma prioridad

Tabla 4. Parámetros admitidos en las clases de la disciplina HTB.

2.4. Clasificación mediante filtros

La clasificación consiste en aplicar reglas a los paquetes para establecer un destino en el árbol jerárquico de disciplinas de la interfaz. Estas reglas se llaman filtros y contienen una serie de condiciones que deben cumplir los paquetes. Éstos filtros se adjuntan a determinados nodos de la jerarquía de disciplinas y forman una lista de filtros. Cada nodo puede contener más de una lista de filtros, cada lista se diferencia por una prioridad representada por un número natural. Así, cuando un filtro se añade a un nodo se debe especificar un parámetro de prioridad, esto hace que el nuevo filtro se añada a una lista donde residen todos los filtros con esa misma prioridad. Cada lista de igual prioridad agrupa filtros en orden secuencial en el mismo orden en que se han añadido.

El proceso de clasificación en un nodo consiste en recorrer las listas de filtros, empezando por la lista más prioritaria. Concretamente, para procesar cada paquete se toma la lista de filtros mas prioritaria, se aplica cada filtro de esta lista secuencialmente, si el paquete cumple todas las condiciones establecidas en un filtro termina el procesado inmediatamente y se redirige el paquete un nodo indicado en el propio filtro. Si se termina una lista y no se cumple ningún filtro se repite el proceso con la siguiente lista de filtros, por orden de prioridad.

La principal restricción de los filtros es que sólo se pueden añadir a las disciplinas *classful*, es decir, las que tienen disponibles clases hijas y el nodo destino del filtro debe ser un nodo de tipo clase. Salvando la restricción anterior, a los nodos restantes del árbol se pueden adjuntar los filtros que se desee, pero los paquetes que llegan a una interfaz de red, son procesados aplicando únicamente por las listas de filtros ubicados en la raíz[1] del árbol. Es decir, el sistema operativo sólo se comunica con el nodo raíz tanto para encolar un paquete como para extraer un paquete para su envío.

El proceso de clasificación se complica si se añaden filtros en diferentes nodos de la jerarquía. De este modo, el sistema de clasificación opera aplicando al paquete entrante los filtros del nodo raíz y estos filtros se resuelven redirigiendo el paquete a un nodo hijo. Si existen filtros en el nodo hijo, se vuelven a aplicar estos filtros para redirigir el paquete a otro nodo hijo, pero el destino debe ser un nodo hijo de este último y así sucesivamente. Este sistema restricciones bastante lógicas para evitar su mal funcionamiento. Como ya se ha indicado, un filtro asociado a un nodo sólo puede redirigir un paquete a uno de sus nodos hijos, es decir, un paquete nunca puede ser clasificado ascendentemente en el árbol.

Esta complejidad en el procedimiento de clasificación persigue como objetivo aumentar la eficiencia del sistema de filtrado. En contrapartida, puede llegar a ser muy complejo de administrar al estar los filtros repartidos por toda la jerarquía. Considere que la eficiencia disminuye enormemente cuando se asocia a un nodo, una lista o listas con muchos filtros, ya que se recorren todos secuencialmente. En esta situación por cada paquete hay que evaluar muchas reglas, pero si se disminuye la cantidad de reglas en un nodo y se distribuyen adecuadamente los filtros entre los nodos hijos, la cantidad reglas aplicadas al paquete hasta alcanzar su nodo destino disminuye considerablemente.

Para salvar la complejidad del sistema de filtrado, en este laboratorio se evitará asociar filtros a nodos

1 No es del todo cierto, si el nodo raíz es *classless*, se considera nodo raíz de los filtros aquel nodo *classful* más cercano a la raíz del árbol.

diferentes del nodo raíz, los ejemplos de clasificación serán lo suficientemente sencillos como para no requerir optimización.

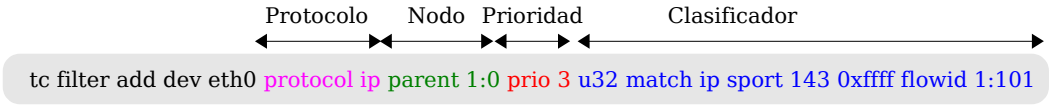

Figura 4. Ejemplo de sintaxis de filtro.

En la figura 4 se presenta con un ejemplo la sintaxis de los filtros, las partes indicadas tienen el siguiente significado:

- Protocolo: Indica el protocolo al que se aplicará el filtro, sólo se contemplará protocolo IP.
- Nodo padre: Nodo de la jerarquía al que está asociado el filtro.
- Prioridad: Indica la lista de prioridad donde se añadirá el filtro. Este número sirve para establecer la preferencia en la evaluación de los filtros, primero se evalúan las listas de filtros con un número menor, si el paquete no se clasifica entonces se proceden con las siguientes listas en orden ascendente. La prioridad es un número natural por ello es 1 el más prioritario y no considera el 0.
- Clasificador: El clasificador indica las comprobaciones a realizar al paquete y el nodo destino si el clasificador es positivo.

En la especificación de un filtro la parte más compleja es el clasificador, por existir diversidad y tener cada uno su propia sintaxis. Los clasificadores más utilizados son dos: *fw* que basa la decisión en netfilter y *u32* capaz de analizar los campos de los paquetes a nivel de bits.

Este laboratorio se centrará en el clasificador *u32* ya que resuelve un amplio conjunto de situaciones de uso común. Otros clasificadores se pueden consultar en la documentación LARTC aunque muchos no están ampliamente documentados.

2.4.1. Clasificador u32

El clasificador *u32* es el más simple de comprender y usar pero su modo de operación es de bajo nivel. Con él se resuelven cantidad de posibilidades ya que selecciona partes del paquete aplicando máscaras a su contenido, aunque presenta cierta dificultad a la hora de interpretar las reglas que lo componen puesto que se requiere el conocimiento exacto de los campos existentes del paquete que se está analizando.

Este clasificador está formado por una lista de selectores, y cada selector es un patrón que debe cumplirse en algún campo del paquete. Si alguno de los selectores se evalúa negativo entonces todo el filtro se evalúa negativo. Cuando todos los selectores se cumplen en su totalidad el filtro se aplica. La sintaxis general es:

```
u32 LISTA_SELECTORES flowid M:N
```

El parámetro final *flowid* es el nodo destino del árbol si el filtro se evalúa positivo y la lista de selectores puede contener tantos selectores como se deseen. Existen tres tipos de selectores y cada uno de ellos sirve para buscar un patrón en el paquete de diferente número de bits. La sintaxis general es la siguiente:

```
match u32 PATRON MASCARA at [DESPLAZAMIENTO | nexthdr+DESPLAZAMIENTO]
match u16 PATRON MASCARA at [DESPLAZAMIENTO | nexthdr+DESPLAZAMIENTO]
match u8  PATRON MASCARA at [DESPLAZAMIENTO | nexthdr+DESPLAZAMIENTO]
```

El significado de los parámetros en las tres posibilidades es el siguiente:

- u32, u16 y u8: Indican el número de bits a comprobar siendo 32, 16 y 8 respectivamente.
- Patrón: es un dato escrito en decimal o hexadecimal que debe coincidir dentro del paquete.
- Máscara: son los bits del patrón de deben coincidir.
- Desplazamiento: posición en bytes dentro del paquete donde se está haciendo la comprobación.

Seguidamente se muestran algunos ejemplos de filtros para ilustrar la sintaxis. Para interpretarlos correctamente se han incluido las figuras 6 y 7 que representan los paquetes IP con sus campos y la cabecera para el protocolo TCP también con sus correspondientes campos.

El primer ejemplo clasifica los paquetes cuyo valor TTL sea 64 al nodo 1:4:

```
tc filter add dev eth0 parent 1:0 prio 10 u32 match u8 64 0xff at 8 flowid 1:4
```

Fíjese en la figura 6 como el tamaño del campo TTL es de 8 bits y el desplazamiento en bytes del campo TTL dentro de la cabecera IP es 8.

Ahora con el siguiente ejemplo se pretende analizar los campos de un protocolo de nivel de transporte como es TCP. El sistema de filtrado facilita la sintaxis mediante *nexthdr* para simplificar el desplazamiento, este parámetro coincide en valor con el tamaño de la cabecera IP. Así, sólo hay que indicar el desplazamiento dentro de la cabecera TCP. El ejemplo mostrado contiene ahora dos condiciones que deben cumplirse: el protocolo debe ser TCP (número 6) y el bit de ACK debe estar activo:

```
tc filter add dev eth1 parent 1:0 prio 10 u32 \
    match u8 6 0xff at 9 \
    match u8 0x10 0x10 at nexthdr+13 \
    flowid 1:3
```

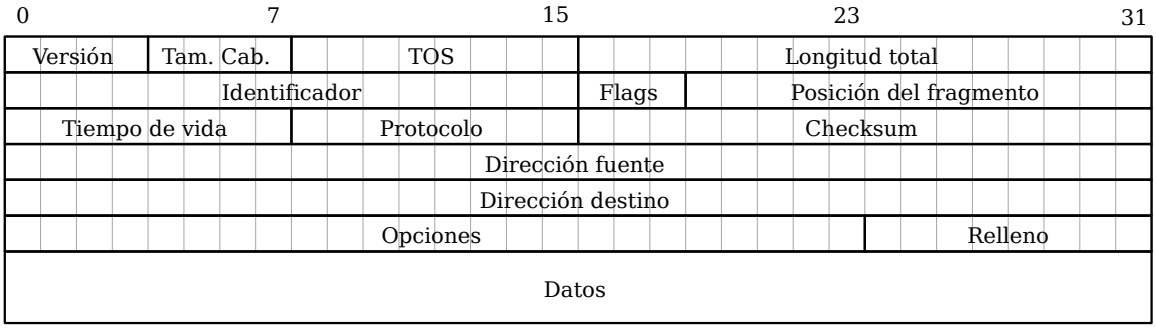

Figura 5. Campos de un paquete IP.

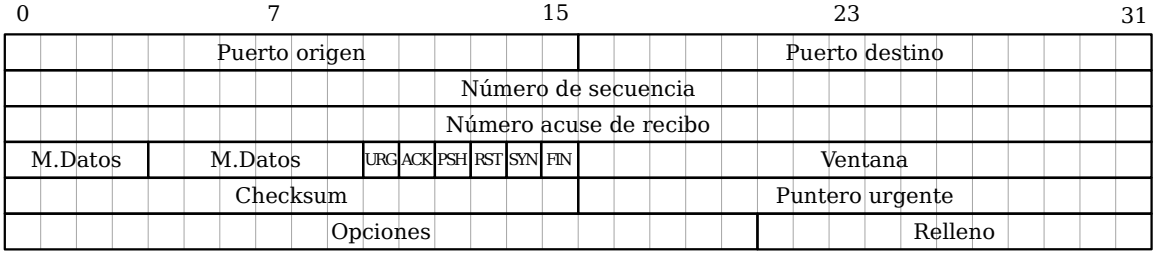

Figura 6. Campos en la cabecera de TCP.

Para facilitar el trabajo con los patrones existen algunos selectores específicos con nombre, de forma que no se tiene que especificar el desplazamiento de cada uno de los campos. En la tabla 5 se muestran algunos y sus equivalencias.

Selector	Equivalencia
`match ip protocol tcp`	`match u8 6 0xff at 9`
`match ip protocol udp`	`match u8 17 0xff at 9`
`match ip src 192.168.0.100/32`	`match u32 0xc0a80064 0xffffffff at 12`
`match ip dst 192.168.20.1/24`	`match u32 0xc0a81464 0xffffff00 at 16`
`match dport 21 0xffff`	`match u16 21 0xffff nexthdr + 2`
`match sport 21 0xffff`	`match u16 21 0xffff nexthdr + 0`
`match ip tos 0x10 0xff`	`match u8 0x1 0xff at 1`

Tabla 5. Selectores con nombre de mayor utilidad y sus equivalencias.

En el último ejemplo se muestra el uso de estos selectores específicos:

```
tc filter add dev eth0 parent 1:0 prio 10 u32 \
    match tcp dport 53 0xffff \
    match ip protocol 6 0xff \
    flowid 1:2
```

2.5. Estadísticas

Para depurar la política de control de tráfico configurada, la herramienta *tc* incluye un modo de

operación con el que presenta estadísticas recopiladas por cada disciplina. Las estadísticas recopiladas representan dos tipos de valores: acumulados e instantáneos. Todas las disciplinas recopilan al menos el número de bytes y de paquetes a los que se les ha aplicado dicha disciplina. La recopilación de estadísticas no se produce sólo en las disciplinas, las clases en muchas ocasiones también recopilan información, siendo ésta muy útil para optimizar la configuración o detectar posibles errores.

El siguiente ejemplo muestra las estadísticas de las disciplinas existentes en una configuración de ejemplo tras la ejecución del comando *tc*:

```
# tc -s -d qdisc show dev eth1

qdisc htb 1: root refcnt 2 r2q 10 default 99 direct_packets_stat 0 ver 3.17
    Sent 4539558 bytes 13203 pkt (dropped 0, overlimits 7115 requeues 0)
    backlog 0b 0p requeues 0
qdisc sfq 299: parent 1:99 limit 127p quantum 1514b flows 127/1024 divisor 1024 perturb 5sec
    Sent 707131 bytes 8123 pkt (dropped 0, overlimits 0 requeues 0)
    backlog 0b 0p requeues 0
qdisc pfifo 222: parent 1:22 limit 100p
    Sent 3603227 bytes 3555 pkt (dropped 0, overlimits 0 requeues 0)
    backlog 0b 0p requeues 0
qdisc sfq 223: parent 1:23 limit 127p quantum 1514b flows 127/1024 divisor 1024 perturb 10sec
    Sent 72894 bytes 193 pkt (dropped 0, overlimits 0 requeues 0)
    backlog 0b 0p requeues 0
```

Los valores estadísticos son fácilmente interpretables: bytes enviados, paquetes enviados y paquetes descartados. El valor *overlimits* indica la cantidad de veces que se podía haber enviado un paquete pero la disciplina rechazó el envío por sobrepasar sus límites establecidos, de hecho, sólo la disciplina HTB establece límites de caudal, por eso, en las estadísticas mostradas sólo pueden existir *overlimits* para HTB.

El siguiente comando *tc* muestra como salida las estadísticas recopiladas en las clases:

```
# tc -s -d class show dev eth1

class htb 1:2 root rate 600000bit ceil 600000bit burst 3000b/8 mpu 0b overhead 0b cburst
1599b/8 mpu 0b overhead 0b level 7
    Sent 4941114 bytes 14340 pkt (dropped 0, overlimits 0 requeues 0)
    rate 48040bit 20pps backlog 0b 0p requeues 0
    lended: 1990 borrowed: 0 giants: 0
    tokens: 599562 ctokens: 307890

class htb 1:99 parent 1:2 leaf 299: prio 1 quantum 5000 rate 80000bit ceil 600000bit burst
1600b/8 mpu 0b overhead 0b cburst 1599b/8 mpu 0b overhead 0b level 0
    Sent 769677 bytes 9020 pkt (dropped 0, overlimits 0 requeues 0)
    rate 7768bit 14pps backlog 0b 0p requeues 0
    lended: 8055 borrowed: 965 giants: 0
    tokens: 2279562 ctokens: 307890
class htb 1:21 parent 1:2 prio 0 quantum 1000 rate 25000bit ceil 100000bit burst 1600b/8 mpu
0b overhead 0b cburst 1600b/8 mpu 0b overhead 0b level 0
    Sent 95380 bytes 1023 pkt (dropped 0, overlimits 0 requeues 0)
    rate 192bit 0pps backlog 0b 0p requeues 0
    lended: 894 borrowed: 129 giants: 0
    tokens: 7400000 ctokens: 1850000

class htb 1:22 parent 1:2 leaf 222: prio 1 quantum 5000 rate 400000bit ceil 600000bit burst
```

```
1600b/8 mpu 0b overhead 0b cburst 1599b/8 mpu 0b overhead 0b level 0
    Sent 3987104 bytes 3949 pkt (dropped 0, overlimits 0 requeues 0)
    rate 38256bit 6pps backlog 0b 0p requeues 0
    lended: 3081 borrowed: 868 giants: 0
    tokens: 477500 ctokens: 318328
```

Los valores estadísticos mostrados en las clases HTB del ejemplo incluyen el valor instantáneo *rate* (caudal) expresado en *bits/seg* y *bytes/seg*. Son interesantes los valores *lended* y *borrowed* que representan los bytes prestados a las clases hijas y los bytes pedidos a la clase padre respectivamente.

2.6. Consideraciones adicionales

Adicionalmente para afinar en el cálculo de la latencia de los paquetes se debe profundizar en el funcionamiento interno del núcleo de Linux.

Según la documentación existente sobre la implementación de la capa de red en el núcleo de Linux, cuando un proceso del sistema envía un paquete, el paquete se pone en la disciplina de la interfaz para su envío.

Pero el controlador del medio físico incluye una cola adicional llamada *ring_buffer*, que se debe considerar. El núcleo se comunica únicamente con la disciplina raíz de la interfaz y solicita paquetes siempre que la cola en anillo del driver no esté llena. Sólo se obtienen paquetes de la disciplina raíz si hay hueco en el anillo de envío y el anillo no está parado, este comportamiento se ilustra en la figura 7.

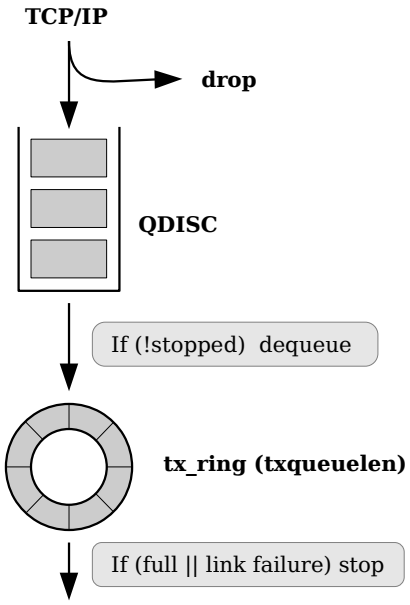

Figura 7. Colas del núcleo de Linux.

El tamaño del anillo de envío es independiente de la cola existente en la disciplina raíz y se puede considerar como una cola adicional inevitable. Este tamaño de anillo puede ser alterado siempre que el driver lo soporte y se puede consultar con el comando *ip link* (parámetro *txqueuelen*). Para cambiarlo sólo es posible mediante el comando *ip link* con la siguiente sintaxis: `ip link set eth0 txqueuelen 500`.

Otra consideración importante es el tamaño MTU cuando se configuran los parámetros de las disciplinas. Es importante ser consciente que muchos parámetros no deben establecerse por debajo de 1500 que es la MTU predeterminada. Por ejemplo, una disciplina TBF o HTB con una cubeta de tamaño inferior a este valor puede hacer que no se envíe ningún paquete, y si se tienen colas en bytes menores de este valor de MTU también puede ocurrir que no se encole ningún paquete.

Por último, considerando el tamaño de MTU es posible configurar una disciplina TBF o HTB para que se comporte como una cubeta con pérdidas (*Leaky bucket*), basta con que el tamaño de la cubeta sea la MTU para que no puedan ocurrir ráfagas.

3. Laboratorio guiado

Las pruebas a realizar en el laboratorio consistirán en realizar el conformado de tráfico para la red virtual desplegada a lo largo de los laboratorios de la asignatura. Se realizará todo el conformado en la máquina que hace de puerta de enlace (GW) simulando que se dispone de un ancho de banda limitado. Todo el control del tráfico se realizará sobre el tráfico saliente, ya que como es habitual, no se puede controlar el tráfico más allá de la frontera de red.

Para limitar este ancho de banda saliente se utilizarán las disciplinas descritas anteriormente, y a lo largo del bloque de laboratorio guiado se realizarán dos ejemplos, el primero consiste en probar diferentes disciplinas y observar los efectos sobre flujos en diferentes condiciones. La segunda parte guiada usará una solución de carácter general basada principalmente en la disciplina HTB donde se comprobará la flexibilidad en la configuración de este tipo de disciplina.

Antes de comenzar se debe preparar el entorno de trabajo con algunos ficheros para poder realizar transferencias de datos por diversos protocolos. Para realizar todas las pruebas se requiere una máquina externa y es recomendable que esté en la misma red, no se recomienda realizar las pruebas desde el equipo local, ya que está conectado mediante una VPN desde una conexión externa con caudal inestable. Así, para este laboratorio se usará en todo momento la máquina virtual compartida.

3.1. Soluciones con múltiples disciplinas

Esta primera parte del laboratorio consiste en probar diferentes disciplinas siguiendo los ejemplos de las sucesivas tareas. Debe guardar todos los comandos *tc* utilizados en los ficheros de texto indicados para su posterior evaluación, recuerde que debe respetar la ubicación y los nombres de los ficheros.

Tarea 1.- Inicie una sesión de escritorio remoto en la máquina compartida para realizar las pruebas de transferencia desde ahí y no cierre la sesión. Simultáneamente en su máquina GW ejecute el comando indicado, como administrador, para mostrar estadísticas sobre la disciplina establecida para la interfaz de forma predeterminada (debe sustituir *eth0* por el nombre de la interfaz externa, si es que difiere).

```
tc -s -d qdisc show dev eth0
```

T1.1.- Cree un directorio nuevo para alojar todos los ficheros de este laboratorio: `mkdir /root/qos`. Igual que en los laboratorios anteriores, debe usarse este directorio y mantener el nombre indicado en cada tarea para una correcta evaluación.

T1.2.- Para facilitar la obtención de estadísticas cree un fichero `/root/qos/estadisticas.sh` y añada el permiso de ejecución al mismo. El fichero mostrará de forma continuada las estadísticas en un terminal, siendo el contenido el siguiente:

```
#!/bin/bash
```

```
watch -n0 '
    tc -s -d qdisc show dev eth0
    echo "-- Clases ------------------------------------"
    tc -s -d class show dev eth0'
```

Código 1. Script para mostrar estadísticas de forma continuada.

T1.3.- Ejecute este *script* (/root/qos/estadisticas.sh) en un terminal y deje este terminal visible durante todas las pruebas realizadas posteriormente.

T1.4.- Para poder realizar pruebas de transmisión debe ubicar un fichero de al menos 20MBytes en tres de las ubicaciones configuradas en la sesión anterior de laboratorio: servidor WEB interno, servidor FTP interno y cuenta de usuario *tai* en GW, esta última es para realizar transferencias por SSH. Puede crear el fichero con el comando *truncate*, el siguiente ejemplo crea un fichero de 20MiB: `truncate -s 20M fichero-grande.dat`.

T1.5.- Ahora hay comprobar que los ficheros creados están ubicados correctamente y son accesibles desde la máquina compartida por diferentes protocolos: HTTP, FTP, y SSH. Esta prueba y todas las demás transferencias se realizan desde la máquina compartida. En primer lugar, para probar HTTP se usará el comando wget de la forma indicada en el código 2 y además, tras la ejecución, debería estar el *fichero-grande.dat* de 20Mbytes en el directorio actual (`ls -lh`), observe en la salida el caudal medio de la transferencia:

```
# wget http://192.168.20.X/fichero-grande.dat

--2020-12-11 19:18:19--  http://192.168.20.X/fichero-grande.dat
Conectando con 192.168.20.X:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 1048576 (1,0M)
Grabando a: “fichero-grande.dat”

fichero-grande.dat     100%[=========================>]   1,00M  --.-KB/s    en 0,002s

2020-12-11 19:18:19 (509 MB/s) - “fichero-grande.dat” guardado [1048576/1048576]
```

Código 2. Comando para realizar transferencias HTTP.

T1.6.- Para probar el servicio FTP de su máquina VM2 se puede utilizar también *wget* escribiendo un URL correspondiente al protocolo FTP. Cuando *wget* usa el protocolo FTP, intenta conectar en modo pasivo al servidor, por ello, debe tener el modo pasivo correctamente configurado del laboratorio anterior. Cree el fichero *fichero-grande.dat* de 20M en la cuenta *tai* de VM2 para que sea accesible por FTP y ejecute la siguiente prueba desde la máquina compartida asegurándose que se ha transferido el fichero:

```
# wget ftp://tai:tai@192.168.20.X/fichero-grande.dat

--2020-12-11 19:48:29--  ftp://tai:*password*@192.168.20.X/fichero-grande.dat
           => “fichero-grande.dat.2”
Conectando con 192.168.20.X:21... conectado.
```

```
Identificándose como tai ... ¡Dentro!
==> SYST ... hecho.   ==> PWD ... hecho.
==> TYPE I ... hecho.  ==> no se necesita CWD.
==> SIZE fichero-grande.dat ... 1048576
==> PASV ... hecho.   ==> RETR fichero-grande.dat ... hecho.
Longitud: 1048576 (1,0M) (probablemente)

fichero-grande.dat. 100%[===================>]   1,00M  --.-KB/s    en 0,001s

2020-12-11 19:48:29 (853 MB/s) - “fichero-grande.dat” guardado [1048576]
```

Código 3. Comando para realizar transferencias FTP.

T1.7.- Por último, para probar una transferencia SSH puede usar el comando *sftp* o *scp*, ambos funcionan a través de un túnel SSH. El siguiente ejemplo se transfiere de nuevo *fichero-grande.dat* por SSH, pero debe crear este fichero en la cuenta *tai* de la máquina GW, ya el servicio SSH no está redirigido a ninguna máquina interna:

```
# sftp tai@192.168.20.X:fichero-grande.dat

tai@192.168.20.X's password:
Connected to 192.168.20.X.
Fetching /home/tai/fichero-grande.dat to fichero-grande.dat
/home/tai/fichero-grande.dat                  100% 1024KB  90.3MB/s   00:00
```

Código 4. Comando para realizar transferencias SSH-SFTP.

Tras probar las transferencias y verificar que el fichero se descarga correctamente (usando los 3 protocolos), se procederá a probar diferentes disciplinas. En todas las tareas posteriores se volverán a realizar las transferencias y medir el caudal usando los tres comandos anteriores.

Tarea 2.- Con la primera disciplina se pretende limitar el caudal de salida de la interfaz exterior del entorno virtual a 300Kbit/s mediante una cubeta con fichas. Ésta disciplina correspondiente a TBF en Linux.

T2.1.- Cree un nuevo fichero `/root/qos/qos-t2.sh` con permiso de ejecución para ir añadiendo los comandos. Con el primer comando se establecerá una disciplina TBF como disciplina raíz de la interfaz, añada en el fichero los comandos indicados:

```
#!/bin/bash
set -x
tc qdisc del dev eth0 root
tc qdisc add dev eth0 root handle 1:0 tbf rate 300Kbit latency 50ms burst 1540
```

Código 5. Ejemplo de disciplina TBF.

T2.2.- Ejecute el script `/root/qos/qos-t2.sh` y observe los efectos en el terminal donde se muestran las estadísticas.

T2.3.- Desde la máquina compartida conéctese al servidor WEB y realice una transferencia mediante HTTP observando la tasa de transferencia, puede usar el comando `wget`

`http://192.168.20.X/fichero-grande.dat`. ¿Corresponde con el caudal de 300Kbit/s establecidos?

T2.4.- Sin detener la transferencia HTTP realice otra transferencia simultánea mediante FTP. Puede utilizar *filezilla* ya que muestra la tasa de transferencia o de nuevo *wget* siguiendo el ejemplo mostrado en el código 3 (pág. 22) ¿Se distribuye el caudal disponible equitativamente entre las dos descargas?

Junto con el material de laboratorio se adjunta un script llamado *descargas.sh* pensado para realizar varias transferencias HTTP/FTP simultáneas. Para que opere correctamente debe usarlo desde el escritorio de una máquina Linux con los programas *pv*, *netcat, socat* y *xterm* instalados (puede instalarlos con APT). Se debe usar en las tareas posteriores y su uso consiste en ejecutarlo con dos argumentos según la siguiente sintaxis: `./descargas.sh N URL` donde N es el número de transferencias simultáneas y URL es la dirección completa del fichero. El URL puede ser tanto el ejemplo mostrado en el código 2 para HTTP o en el código 3 para FTP.

Tarea 3.- Descargue en la máquina compartida el script *descargas.sh* y añada el permiso de ejecución al mismo con el comando *chmod*. Ejecute el comando sin argumentos para que se muestre la ayuda de uso.

T3.1.- Desde el escritorio remoto de la máquina compartida se probarán tres transferencias simultáneas hacia su GW. Ejecute el comando indicado utilizando su IP pública y se abrirán 3 ventanas donde debe observar la tasa de transferencia (considere que a veces las ventanas se superponen y tiene que moverlas):

```
./descargas.sh 3 http://192.168.20.X/fichero-grande.dat
```

T3.2.- Repita el comando anterior realizando simultáneamente 10 transferencias HTTP. Sin parar estas últimas, de manera simultánea intente acceder por FTP para descargar archivos, comprobará que las conexiones no funcionan correctamente.

T3.3.- De nuevo, sin detener ninguna transferencia intente acceder por SSH a su GW para obtener un terminal de texto de la máquina. ¿Responde correctamente el flujo interactivo SSH con las pulsaciones de teclas?

En la situación anterior ocurren varios efectos, el principal son las fluctuaciones en la tasa de cada transferencia. Se debe principalmente al descarte de multitud de paquetes de la cola de salida, y el resultado observable habitualmente es que una de las transferencias se apodera de prácticamente todo el caudal disponible dejando detenidas las demás.

Tarea 4.- Se realizará otra mejora creando un árbol de disciplinas jerárquico. Se utilizará la disciplina PRIO con tres bandas de prioridad con el objetivo de dar prioridad al servidor WEB.

T4.1.- Copie el *script* anterior `qos-t2.sh` con el nombre `qos-t4.sh` para añadir una nueva línea con el comando:

```
tc qdisc add dev eth0 parent 1:0 handle 10: prio
```

T4.2.- Ejecute el nuevo *script* (qos-t4.sh) y compruebe, usando el terminal con las estadísticas, si se han añadido correctamente las disciplinas. Deben existir 3 bandas creadas automáticamente, asegúrese que la configuración mostrada corresponde a la figura 8.

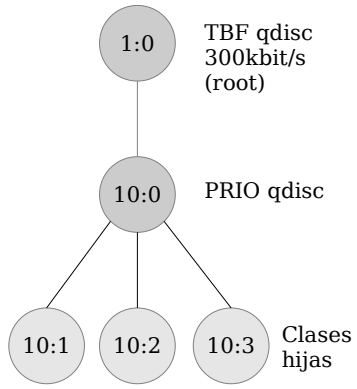

Figura 8. Disciplina PRIO.

T4.3.- Se añadirán dos filtros un para dar prioridad a los paquetes HTTP sobre el resto (banda 1) y otro para enviar el resto del tráfico a la banda 3. Recuerde que los filtros sólo pueden añadirse a la disciplina 10:0 por ser de tipo *classful*. Añada al fichero qos-t4.sh los filtros indicados, siendo cuidadoso en los retornos de carro, en el fichero de *script* no pueden existir espacios entre la barra y el salto de línea. Siguiendo este procedimiento el *shell* interpreta que todo el comando es una única línea.

```
tc filter add dev eth0 \
    protocol ip parent 10: prio 1 u32 \
    match ip sport 80 0xffff flowid 10:1

tc filter add dev eth0 \
    protocol ip parent 10: prio 1 u32 \
    match ip dst 0.0.0.0/0 flowid 10:3
```

T4.4.- Realice una única transferencia HTTP y en el terminal con estadísticas observe si los paquetes HTTP se están clasificando en la clase 10:1, debe observar si aumenta el número de bytes y sólo hay paquetes enviados en esa clase.

T4.5.- Pare la transferencia y conecte con su máquina GW por SSH obteniendo un terminal, pulse teclas para observar en qué clase están clasificados los paquetes del flujo SSH. Sin cerrar la conexión SSH, ejecute en la máquina compartida el script de descargas simultáneas con al menos 5 transferencias HTTP. Cuando estén las conexiones activas, intente interactuar en el terminal SSH con la máquina GW. ¿Es el comportamiento el esperado en estas circunstancias?

T4.6.- En la primera línea del fichero está la cubeta que limita la velocidad a 300Kbits por segundo, aumente el caudal de salida a 1Mbit. Tras cargar el script modificado realice de nuevo 5 descargas simultáneas y al mismo tiempo intente acceder mediante FTP desde la máquina anfitrión. ¿Responde el servidor FTP?

En el ejemplo mostrado la cola prioritaria anula completamente el resto de flujos menos prioritarios, la planificación es estricta, mientras una cola prioritaria tenga paquetes encolados, sólo se extraen paquetes de esta cola. Otro aspecto importante que se debe tratar es el modo en que se aplican los filtros. En la sección 2.4 (pág 14) se indicó que los filtros se agrupan listas ordenadas y cada lista ordenada tiene una prioridad, de forma que, el orden de aplicación es recorrer cada lista hasta que un filtro se aplique. En cuanto un filtro se aplique, se para el procesamiento de filtros en ese nodo de la jerarquía.

Tarea 5.- Para entender mejor el funcionamiento interno de los filtros realice los cambios indicados

sobre un nuevo fichero llamado `qos-t5.sh` partiendo de una copia el anterior `qos-t4.sh`.

T5.1.- En el nuevo fichero invierta el orden de los filtros, debería quedar como sigue:

```
tc filter add dev eth0 \
    protocol ip parent 10: prio 1 u32 \
    match ip dst 0.0.0.0/0 flowid 10:3

tc filter add dev eth0 \
    protocol ip parent 10: prio 1 u32 \
    match ip sport 80 0xffff flowid 10:1
```

T5.2.- Cargue y pruebe a realizar una transferencia HTTP y observe el terminal de estadísticas. ¿Se clasifican los paquetes con puerto origen 80 en la clase 10:1? ¿Por qué nunca se aplica el segundo filtro?

T5.3.- Sin cambiar el orden de los filtros en el fichero, debe conseguir que el tráfico se clasifique del mismo modo que en la tarea anterior. Para hacerlo cambie la prioridad de cada filtro y considere que la mínima prioridad permitida es 1, no se puede poner 0.

Tarea 6.- Ahora se propone añadir filtros para dar prioridad a los flujos SSH, ya que suelen ser interactivos y habitualmente no consumen gran ancho de banda.

T6.1.- Copie el fichero `qos-t5.sh` como `qos-t6.sh` y realice dos cambios: (1) cambie el filtro HTTP para que estos flujos se clasifiquen en la clase 10:3 y (2) añada un nuevo filtro para que los flujos SSH se clasifiquen en la clase 10:1 (SSH usa el puerto 22).

T6.2.- Ejecute el nuevo *script* `qos-t6.sh` y realice 10 transferencias simultáneas HTTP. Al mismo tiempo compruebe que sigue respondiendo el servidor SSH de manera interactiva en un terminal.

T6.3.- Sin detener las 10 descargas proceda a transferir por SSH el fichero de gran tamaño mediante el comando *scp* o *sftp* siguiendo el ejemplo del código 4 (pág. 22) ¿Deja el terminal SSH de responder interactivamente? ¿Siguen operando correctamente las transferencias HTTP simultáneas?

T6.4.- Sin detener ninguna transferencia, intente bajo estas circunstancias usar una de las máquinas internas para conectarse a Internet. Por ejemplo, desde máquina interna VM1 actualice el listado de los paquetes con `apt update`, seguramente observará que toda la red interna ha perdido la conexión con la red exterior, el enlace de salida está copado por la transferencia SSH-SFTP.

Tras analizar las pruebas realizadas en las últimas tareas han surgido los siguientes problemas:

- Cuando varios flujos compiten en una cola siempre alguno de ellos es dominante sobre los demás, esto se ha visto al realizar multitud de transferencias simultáneas HTTP, todas suelen detenerse menos una.
- Los flujos clasificados en la cola de mayor prioridad anulan el resto de colas, es decir, acaparan todo el caudal disponible. Esta situación empeora en un ataque DoS hacia el servicio prioritario: se anularían todos los flujos, incluso queda fuera de servicio la red interna al no poder ni resolverse las solicitudes DNS.

Para atajar en la medida de lo posible esta situación se ampliará el árbol de disciplinas en las siguientes tareas.

Tarea 7.- En primer lugar se intentarán equilibrar los caudales de las transferencias simultáneas HTTP. Se añadirá una política SFQ en el árbol para tratar los flujos HTTP tal y como se muestra en la figura 9. Esta política es similar a WFQ explicada en teoría pero aplicada a flujos detectados dinámicamente.

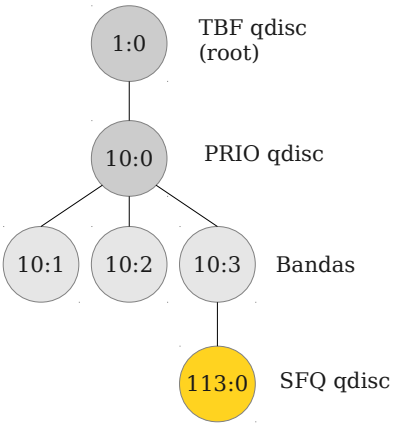

Figura 9. Disciplina SFQ.

T7.1.- De nuevo copie el fichero qos-t6.sh como qos-t7.sh y añada a este último la nueva disciplina. Añádala antes de los filtros, no al final del fichero, de la siguiente forma:

```
tc qdisc add dev eth0 parent 10:3 handle 113: sfq
```

T7.2.- Pruebe desde la máquina compartida 10 transferencias simultáneas. Observe si todos los flujos adquieren ancho de banda de una manera más equitativa.

T7.3.- Observe en la máquina GW la ventana de estadísticas, deberán ir apareciendo clases hijas dinámicamente en el nodo 10:3 por cada uno de los flujos detectados. Cuando paren las transferencias estas clases desaparecerán.

Tarea 8.- Vuelva a copiar el script de la tarea anterior como qos-t8.sh, añada un filtro para el protocolo FTP con el objetivo de establecer la clasificación indicada en la figura 10.

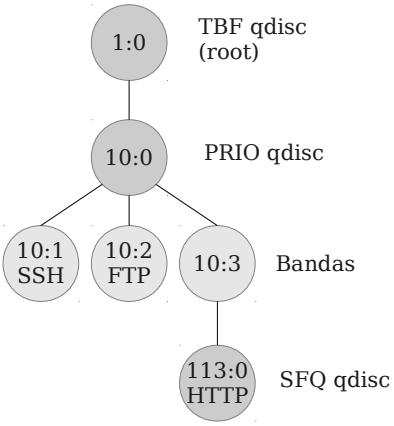

Figura 10. Ubicación de los flujos.

T8.1.- Ahora debe eliminar el filtro de T5.1.- que clasificaba todo los paquetes con destino 0.0.0.0/0. A cambio, se desea clasificar el tráfico TCP restante en la clase 10:2 mediante el siguiente filtro (fíjese en la prioridad del filtro):

```
tc filter add dev eth0 \
    protocol ip parent 10: prio 99 u32 \
    match ip protocol 6 0xff \
    flowid 10:2
```

T8.2.- Ejecute 2 transferencias simultáneas HTTP, y una vez iniciadas, comience una transferencia FTP simultánea. ¿Por qué se paran las dos transferencias HTTP? Compruebe si el servicio SSH sigue operando con prioridad en estas circunstancias.

T8.3.- Detenga la transferencia FTP e inicie 10 transferencias HTTP verificando que las máquinas internas no han perdido la conexión a Internet. ¿Por qué ahora la sobrecarga HTTP no afecta a la red interna?

T8.4.- En T4.3.- se añadió un filtro encargado de clasificar cualquier paquete IP en la banda 10:3, en cambio ahora en T8.1.- el nuevo filtro a pesar de estar después del anterior, tiene prioridad ¿por qué

ocurre esto?. Considere que este tipo de filtros no operan exactamente igual que *nftables* en el cual sólo era importante el orden de las reglas. ¿En qué banda se clasifican por tanto los paquetes UDP?

La solución anterior todavía presenta algunos problemas ya que pueden ocurrir situaciones donde el servidor HTTP se quede sin servicio, principalmente si la clase 10:2 se satura debido al tráfico de los equipos de la red interna. Además, habrá observado que las 10 transferencias simultáneas no se sirven adecuadamente, se probarán más soluciones a continuación.

Tarea 9.- En primer lugar se pretende evitar que la cola 10:2, establecida como predeterminada, anule el tráfico HTTP. Así, se le añadirá una disciplina TBF con un caudal máximo inferior al disponible, por ejemplo 800Kbits, quedando al menos 200Kbits para la siguiente cola menos prioritaria.

T9.1.- De nuevo copie el fichero de la tarea anterior como `qos-t9.sh` para trabajar en él. Añada una disciplina TBF en el árbol hijo de 10:2 de la siguiente forma:

```
tc qdisc add dev eth0 parent 10:2 handle 112: \
   tbf rate 800kbit limit 10k burst 3000
```

T9.2.- Ejecute el script con la nueva disciplina, inicie transferencias HTTP y algunas FTP simultáneamente. Compruebe si las transferencias HTTP siguen operando pero con menor caudal.

Tarea 10.- Termine la solución en un fichero llamado `qos-t10.sh` siguiendo el esquema de figura 11. Debe realizar una clasificación de los paquetes DNS (protocolo UDP) y añadir una disciplina SFQ a la cola prioritaria como la indicada.

T10.1.- Una vez implementada realice 20 ó 30 transferencias HTTP simultáneas ¿se equilibra el caudal de las descargas?.

T10.2.- Ahora en la máquina que ejecuta el servidor HTTP cambie la MTU a 100 mediante `ip link set dev eth0 mtu 100`. Repita las 30 descargas y observe el resultado ¿Por qué ahora si se equilibran las descargas?.

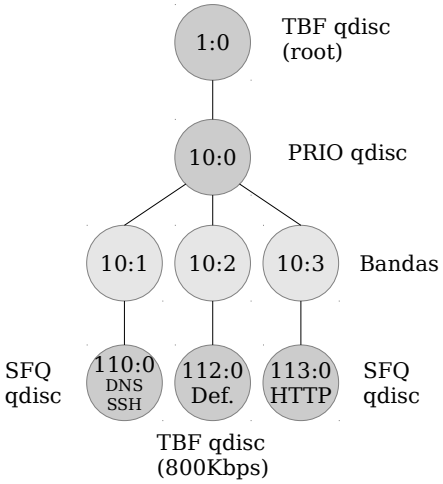

Figura 11. Ejemplo con múltiples disciplinas.

En el último caso mostrado se disminuye el tamaño MTU y el efecto inmediato es una disminución considerable de la eficiencia de los protocolos al aumentar la relación entre los tamaños de las cabeceras y los datos enviados de cada paquete. En cambio, aumenta la interactividad y la fluidez al enviarse más paquetes por segundo. Para comprender lo ocurrido considere que en el ejemplo se dispone de 1Mbit/seg y se deben repartir entre 30 flujos HTTP siendo la MTU de 1500bytes, por tanto, los paquetes son de este tamaño. Haciendo el siguiente cálculo:

$$1\,Mbits/seg = 125000\,Bytes/seg$$

$$\frac{125000\,Bytes/seg}{1500\,Bytes} = 83\,paquetes/seg \quad \Rightarrow \quad \frac{83\,paquetes/seg}{30\,flujos} \simeq 2.7\,paquetes/seg$$

El resultado muestra que cada flujo apenas recibe más de 2 paquetes por segundo, llevando esta situación al extremo se podrían agotar los tiempos de espera de los paquetes TCP. Disminuyendo la MTU a 100Bytes tal y como se hizo en T10.2.- se aumenta el número de paquetes por segundo que recibe cada flujo, llegando a alcanzar 41 paquetes por segundo y aumentando la fluidez. Este procedimiento aumenta la interactividad de un flujo aunque se pierda eficiencia, pero hay que ser cuidadoso al aplicarlo ya que genera otro tipos de problemas de red, ya que pueden descartarse paquetes con MTU superior.

El caso mostrado como ejemplo llega a complicarse si se aumenta el número de servicios y el número de bandas de prioridad. Llega a ser complicado el cálculo de los tamaños de cola, latencias y caudales para cada uno de los nodos, y además de la complejidad, esta solución no aprovecha bien el caudal disponible, fíjese cómo en el flujo predeterminado (nodo 10:2) se ha establecido una disciplina TBF con un caudal de 800Kbps, durante el tiempo en que no existe otro tipo de tráfico sólo se utiliza 800Kbits/s del 1Mbits/s disponible.

Por los motivos expuestos se propone el estudio de soluciones basadas en la disciplina HTB que mejora la eficiencia.

3.2. Implementación con HTB

HTB simplifica el procedimiento de clasificación y aprovecha en mayor medida el ancho de banda disponible. La configuración de este tipo de disciplina es sencilla, de hecho, la página de manual de la misma (`man tc-htb`) es muy reducida ya que, aunque tiene pocos parámetros, éstos permiten muchas posibilidades de configuración. En los ejemplos propuestos se verá como es posible implementar diferentes disciplinas usando únicamente HTB con diferentes configuraciones.

Tarea 11.- En primer lugar se establecerá como disciplina raíz de la interfaz una disciplina HTB. Cree un nuevo fichero `qos-t11.sh` para almacenar la configuración y añada el siguiente contenido:

```
#!/bin/bash
set -x
tc qdisc del dev eth0 root
tc qdisc add dev eth0 root handle 1:0 htb default 1
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 1Mbit burst 1500
```

T11.1.- Realice una transferencia HTTP y pruebe si opera correctamente el límite establecido. Después, realice 10 transferencias simultáneas HTTP para comprobar si se equilibra la velocidad entre diferentes descargas o funcionan irregularmente como en casos anteriores.

T11.2.- Use la ventana de estadísticas para ver como HTB muestra los paquetes y los tokens además de otros datos.

En el ejemplo anterior se ha creado una disciplina raíz del tipo HTB en la interfaz, pero es obligatorio indicar cual es la clase hija donde se encolará de manera predeterminada el tráfico, para el ejemplo, corresponde a `default 1`, indicado en rojo. El número indicado en el parámetro *default* es el número

menor de la clase hija destino, es decir `default 1` clasifica los paquetes a la clase hija `1:1`. La configuración HTB realizada en el ejemplo es equivalente a la realizada en la sección anterior mediante TBF para limitar todo el caudal de la interfaz.

Ahora se presenta la estructura jerárquica de HTB y el proceso *Round-Robin* seguido en el reparto del caudal. Se propone realizar una configuración usando únicamente en HTB que intente asemejarse a una disciplina PRIO (colas de prioridad). Observe el parecido entre la figura 8 (pág. 24) y lo mostrado en la figura 12. En esta última se establecen prioridades a las clases HTB consiguiendo el un efecto parecido a una disciplina PRIO, pero presenta algunas diferencias que se observarán a continuación en el ejemplo usado.

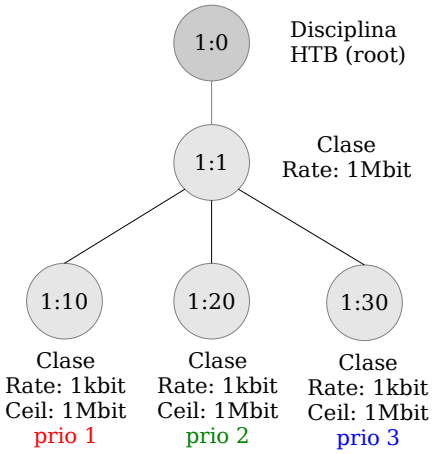

Figura 12. Jerarquía HTB equivalente a PRIO.

En primer lugar, se utilizará la característica de caudal mínimo/máximo de HTB y se observará como la clase padre distribuye ancho de banda entre los hijos. Para comprender el siguiente ejemplo, recuerde que el parámetro *rate* establece el caudal garantizado y el parámetro *ceil* es el máximo caudal alcanzable por la clase. En el proceso de reparto de caudal entre los hijos, la prioridad establece quien es el primer hijo que recibe el ancho de banda sobrante, y cuando no quede caudal sobrante, los menos prioritarios no reciben nada.

Tarea 12.- Para implementar la jerarquía mostrada en la figura 12 use un nuevo fichero llamado `qos-t12.sh`, y como antes, copie `qos-t11.sh` para usarlo como punto de partida.

T12.1.- En el nuevo fichero establezca para la disciplina HTB raíz como clase predeterminada la `1:30`, es decir cambie `default 30` y añada tres clases hijas a 1:1 mediante:

```
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 1kbit ceil 1Mbit prio 1
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 1kbit ceil 1Mbit prio 2
tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 1Mbit prio 3
```

T12.2.- Ahora añada un filtro que clasifique todo el tráfico HTTP hacia la clase `1:10`. Utilice los ejemplos ya mostrados considerando que el filtro debe tener como padre, la raíz del árbol `1:0`.

T12.3.- Desde la máquina compartida inicie una única transferencia HTTP y en la ventana de estadísticas verifique que el filtro anterior opera correctamente, si es así, verá en las estadísticas como se incrementan los bytes enviados de la clase `htb 1:10`.

T12.4.- Manteniendo activa la transferencia HTTP anterior, realice al mismo tiempo una transferencia FTP, puede usar *wget* o *descargas.sh* del modo indicado en T1.6.-. Observará que la transferencia HTTP mantiene estable su tasa de transferencia mientras FTP opera con el caudal mínimo de 1kbit.

T12.5.- Como última prueba, pare todas las transferencias y conecte un terminal por SSH a su GW desde la máquina compartida. Desde otro terminal realice 10 transferencias HTTP simultáneas. Escriba algunas letras en el terminal SSH y ejecute algunos comandos (`ls -l`, `cd`, `mkdir`, etc.), notará

como cada vez responde de forma más lenta hasta que se vuelva prácticamente inusable.

T12.6.- Si compara el ejemplo anterior con el realizado anteriormente pero con las disciplinas PRIO (T4.5.-) el resultado es diferente. Con las colas de prioridad se anulaba SSH completamente, pero en este caso, no se anula del todo. Es debido a que las clases tienen un caudal mínimo de 1kbit (*rate 1kbit*). Para conseguir el mismo efecto que con PRIO habría que poner *rate* a cero ¿Qué ocurre si intenta poner *rate* a cero o a menos de 1kbit en fichero qos-12.sh?

T12.7.- Como última prueba, realice 20 o más transferencias HTTP para observar el desequilibrio en los caudales. Soluciónelo añadiendo una disciplina SFQ a la clase 1:10 y reduciendo el tamaño MTU de la máquina que sirve las peticiones HTTP.

En el ejemplo mostrado se ha conseguido un efecto parecido, pero no igual, que el de las colas de prioridad. Se ha utilizado el mínimo *rate* permitido para las clases hijas, y se ha establecido adecuadamente el parámetro *prio* de cada una de las clases. El parámetro *prio* hace referencia a la prioridad en el reparto del caudal disponible desde la clase padre y el parámetro *ceil* indica el máximo caudal que una clase toma cuando la clase padre reparte caudal. Con todo esto, el funcionamiento se puede resumir como sigue:

La clase padre 1:1 dispone de un caudal de 1Mbit para repartir entre las clases hijas. Cuando las clases hijas solicitan caudal, pueden solicitar a la clase padre hasta 1Mbit (parámetro *ceil 1Mbit*). El algoritmo de reparto de la clase padre es el siguiente:

1. Se ordena por prioridad a todos los hijos que solicitan caudal.
2. Al más prioritario se le asigna todo el caudal solicitado sin sobrepasar el caudal disponible (1Mbit).
3. Si queda caudal disponible (no se ha sobrepasado 1Mbit), se vuelve aplicar el paso 1 con los hijos restantes y se hace el reparto de nuevo del paso 2 con el caudal disponible. Y así hasta agotar el caudal disponible.

Aplicando el algoritmo anterior al ejemplo de la figura 12, al solicitar el primer hijo caudal, puede quedarse con todo el caudal disponible del padre (1Mbit) ya que es el más prioritario (*prio 1*). Así, el resto de hijos sólo tienen 1kbit de ancho de banda.

La configuración mostrada no es equivalente a las colas de prioridad puesto que no es posible establecer el parámetro *rate* a cero en HTB. Además, cada clase HTB tiene parámetros que controlan las ráfagas (*burst* y *cburst*) que provocan otro efectos, como el mostrado en T12.5.- donde el terminal SSH responde inicialmente a las pulsaciones hasta agotar los tokens/bytes de ráfaga y dejar de responder totalmente tras varias pulsaciones de teclas continuadas.

HTB está pensado para garantizar caudal a todas las clases, ese es el principal motivo por el que no permite establecer *rate* a cero. En este sentido, el uso habitual de HTB consiste en dotar a cada una de las clases de un caudal garantizado (*rate*) suficiente para que el servicio, aunque lento, siga operando. Siguiendo este principio se propone usar una configuración razonable para mejorar considerablemente el funcionamiento del ejemplo mostrado anteriormente y, persiguiendo también el objetivo de

aprovechar el ancho de banda completo disponible.

Tarea 13.- Siguiendo el esquema de la figura 13 y trabajando en un nuevo fichero de nombre qos-t13.sh cambie los parámetros *rate* de las tres clases 1:1X y establezca el parámetro *ceil* al valor indicado.

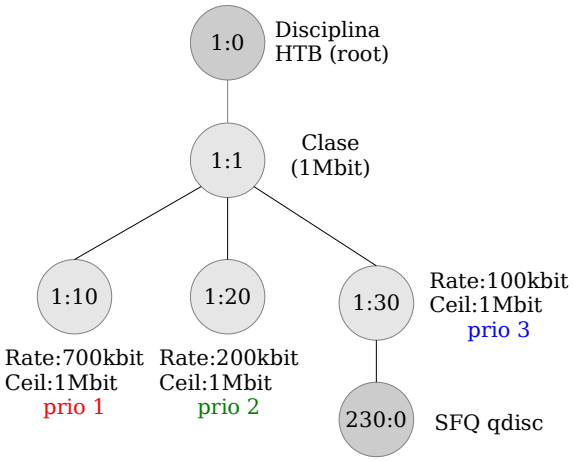

Figura 13. Jerarquía HTB con disciplinas adicionales.

T13.1.- Clasifique todo el tráfico de la siguiente forma:

- Clase predeterminada 1:20
- Tráfico HTTP en la clase 1:30.
- Tráfico SSH en 1:10.
- Cambie la disciplina SFQ a la clase 1:30.

T13.2.- Realice 20 transferencias simultáneas HTTP pruebe si la conexión SSH responde de manera interactiva.

T13.3.- Anule las 10 transferencias y realice una única transferencia HTTP para medir el caudal. Al mismo tiempo realice otra transferencia FTP ¿garantiza la clase 1:30 100kbits al tráfico HTTP? ¿Cuál es el caudal teórico que debe usar la conexión FTP? ¿Coincide el caudal teórico disponible para FTP en esta situación?

Comparando esta solución HTB con la solución multidisciplinar de la sección anterior se llega a la conclusión que HTB ofrece mayor facilidad y flexibilidad de configuración en su uso y configuración.

4. Estudio no guiado

Para finalizar el laboratorio se propone realizar un ejercicio no guiado usando una jerarquía con la disciplina HTB. Concretamente, el ejemplo mostrado en la figura 14 corresponde a una posible configuración para un caudal de salida de un servidor con diferentes servicios, donde sólo se controla la frontera de red. Realice la implementación según las indicaciones siguientes:

Tarea 14.- Implemente la jerarquía HTB mostrada en la figura 14 en un nuevo fichero `/root/qos/qos-t14.sh` y siguiendo las siguientes consideraciones.

T14.1.- La clase 1:1 debe incluirla pero no se usará en esta tarea, se usará posteriormente.

T14.2.- En el filtro para DNS considere que el servicio DNS se refiere a la peticiones DNS realizadas a servidores externos.

T14.3.- El filtro para el protocolo SMTP se refiere a todos los paquetes salientes desde el puerto 25.

T14.4.- La clase 1:99 es la clase predeterminada para todo el tráfico restante.

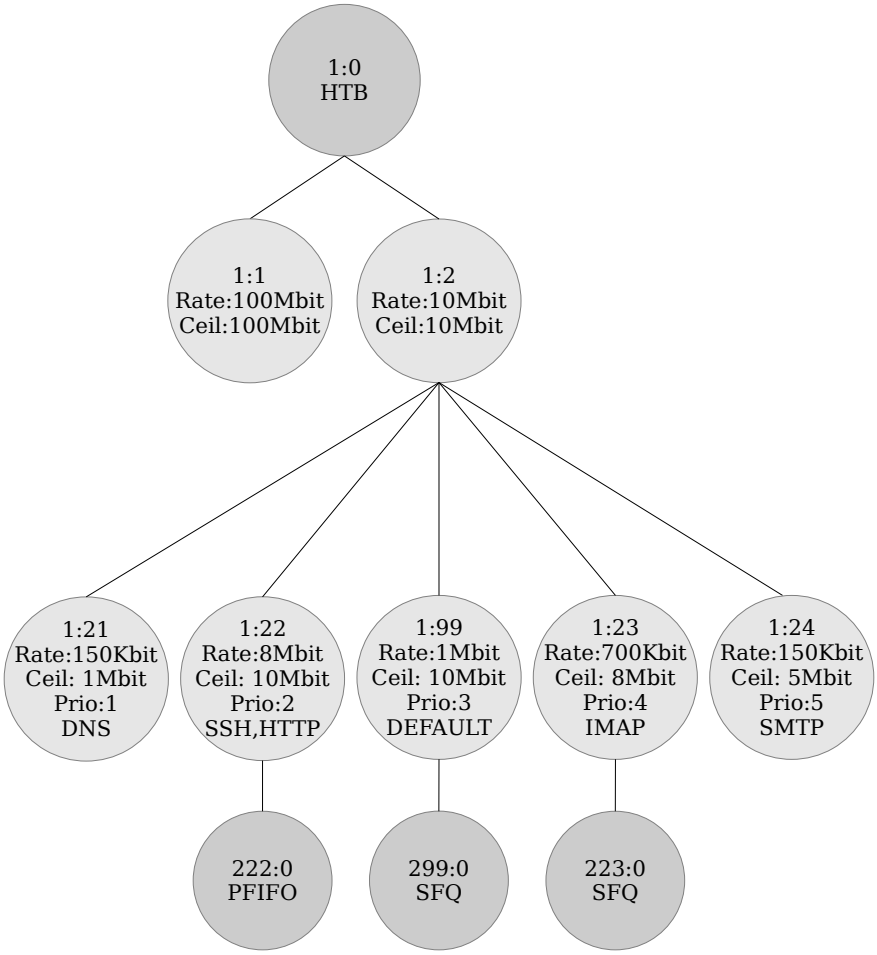

Figura 14. Ejemplo de jerarquía HTB con múltiples servicios.

En la tarea anterior no se usó la clase 1:1 la cual tiene asignada un caudal de 100Mbits. Ésta clase está pensada para dar servicio con mayor ancho de banda a toda la red virtual 192.168.20.0/24, pero para realizar las pruebas se necesita un equipo exterior a la red 192.168.20.0/24 con un ancho de banda estable. Por ello se deberían realizar las pruebas desde los equipos del laboratorio, ya que al intentarlo con la VPN de forma remota los resultados no serán estables.

Alternativamente, para poder testar las disciplinas desde la máquina compartida, se propone usar la configuración de la tarea anterior, donde no hay ningún filtro para la red 192.168.20.0/24. Una vez verificada la configuración anterior se añadirá un filtro para clasificar la red virtual en la clase 1:1. Siga las siguientes indicaciones para realizar las pruebas desde la máquina compartida.

Tarea 15.- Para realizar las pruebas debe utilizar los scripts *escucha-puerto.sh* y *descargas2.sh*. El primero se usará en la máquina GW y el segundo en la máquina compartida.

T15.1.- Instale el paquete *socat* con *apt* en la máquina GW y pruebe el script *escucha-puerto.sh* en un terminal de forma que se ponga a esperar conexiones en el puerto de correo electrónico (SMTP).

T15.2.- Desde la máquina compartida conecte con usando el comando `nc 192.168.20.X smtp` y vea

lo que recibe por el terminal. Este flujo de datos es infinito así que debe usar CTRL+C para cortar la conexión.

T15.3.- También desde la máquina compartida ejecute el script *descargas2.sh* con varias descargas simultáneas para testar el puerto de correo SMTP.

T15.4.- Use adecuadamente el script *escucha-puerto.sh* para testar también el puerto IMAP.

T15.5.- Con la configuración utilice *descargas.sh* y *descargas2.sh* masivamente hacia diferentes puertos y compruebe que opera correctamente su configuración.

T15.6.- En la situación previa de estrés asegúrese que la máquina VM1 y VM2 son capaces de resolver peticiones DNS desde la red interna sin ninguna demora, y que pueden también actualizar la lista de paquetes o hacer instalaciones de software con APT.

Para finalizar el ejercicio se añadirá un filtro para acelerar la red local/virtual, reutilizando el fichero anterior y añadiendo sólo un filtro para la red indicada

Tarea 16.- Cree un nuevo script vacío llamado `/root/qos/qos-t16.sh` de forma que incluya el script de la tarea anterior. El contenido podría ser de la siguiente forma:

```
#!/bin/bash

# Esta línea incluye otro fichero
source /root/qos/qos-t14.sh

# A partir de aqui se pueden añadir sentencias adicionales
```

T16.1.- Tal y como se muestra en el comentario del código anterior, al final del fichero se pueden añadir sentencias. Añada al final del nuevo fichero un filtro prioritario que clasifique todo el tráfico de la red local-virtual en la clase 1:1. Use una regla prioritaria para clasificar todos los paquetes con destino la red 192.168.20.0/24 en esta clase.

T16.2.- Ahora, para verificar su funcionamiento, compruebe el caudal de una transferencia HTTP desde la máquina compartida.

T16.3.- Para asegurarse que para el resto de redes sigue limitando el caudal, desde su equipo local realice una transferencia SMTP y conecte por SSH al mismo tiempo para ver en las estadísticas si la clasificación es correcta.

T16.4.- Considere que debe añadir un servicio VPN considerado como prioritario ¿A qué clase lo añadiría?, añada un filtro que lo clasifique según su decisión.

Tarea 17.- Opcional-5a: Puede realizar algunas pruebas opcionales no estudiadas en este laboratorio

T17.1.- Experimente con la disciplina CHOKE colapsando algún servicio. ¿Para qué sirve esta disciplina?

T17.2.- Compruebe que SFQ no es buena política para páginas WEB, si la página tiene mucho contenido (muchas imágenes) ocurren efectos extraños, se cargan todas la imágenes en paralelo y lentamente. Puede probar una simple PFIFO con un tamaño de un solo paquete.

T17.3.- Pruebe alterar el tamaño de cola de la interfaz para ver si afecta a los resultados, establezca valores grandes y pequeños con el comando *ip* y altere el tamaño de ráfaga en la clase 1:20.