Control de Tráfico en Linux

Sistemas Telemáticos para Medios Audiovisuales

Departamento de Teoría de la Señal y Comunicaciones y
Sistemas Telemáticos y Computación

Noviembre de 2018

Contenidos

Contenidos

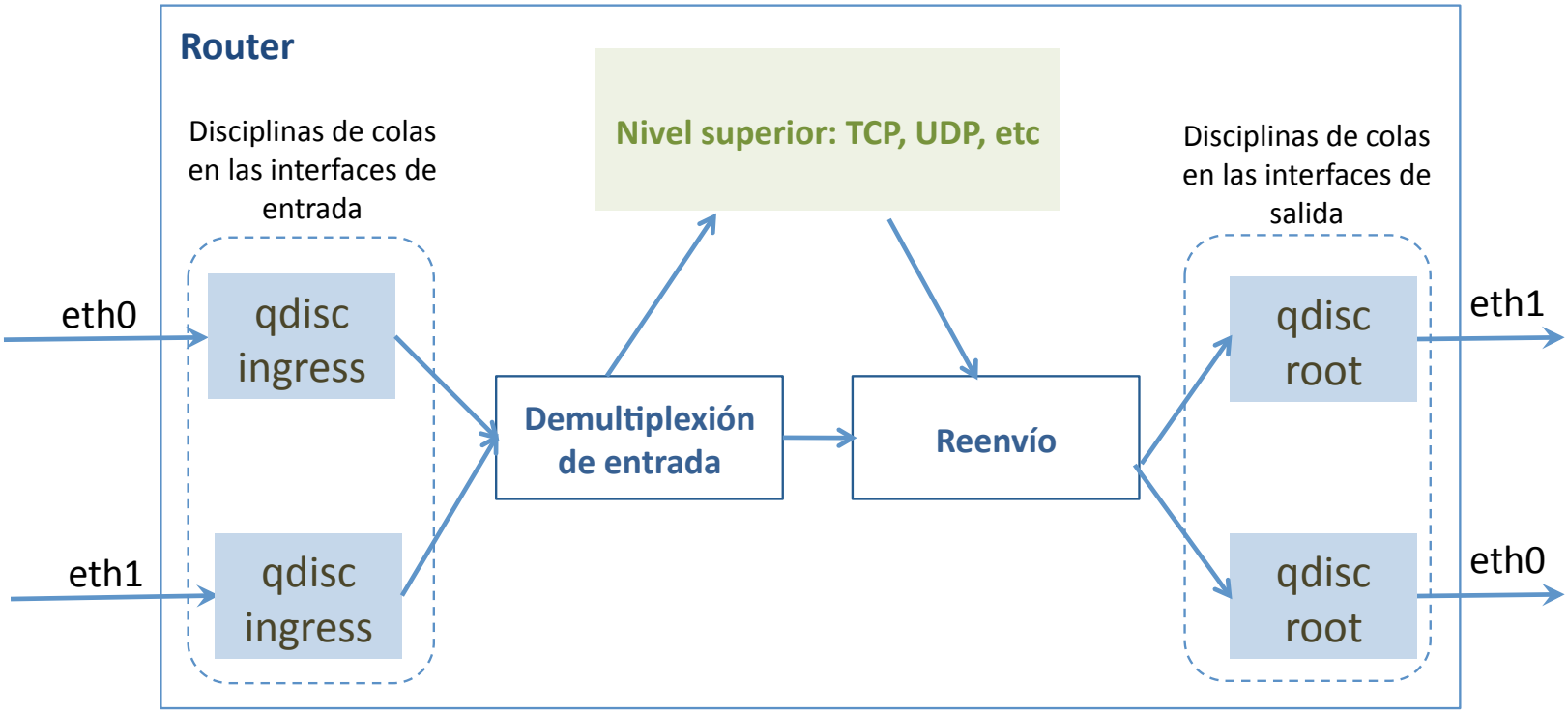
Control de tráfico en Linux

- tc (*traffic control*) es una herramienta que permite realizar el control de tráfico en Linux.
- tc utiliza disciplinas de colas, denominadas `qdisc` (*queueing discipline*).
- Cada vez que el kernel tiene que enviar un paquete a una interfaz, se encola en la `qdisc` configurada en dicha interfaz. El kernel trata de sacar el máximo número de paquetes de dicha `qdisc` para entregárselos al driver de la tarjeta de red.
- El tipo de `qdisc` más simple es FIFO (First In First Out). Si el driver de la tarjeta de red está ocupado, el paquete se quedará guardado en la cola.

Contenidos

Interfaces de entrada/salida de un router

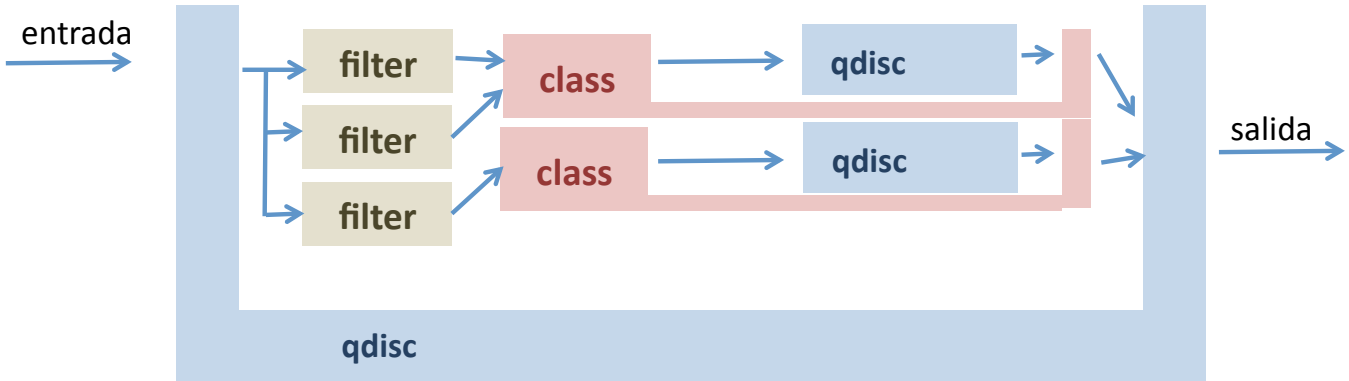
- El control de tráfico se puede aplicar en la interfaz de entrada de un router o en la interfaz de salida del router.
- Cada interfaz de red de un router puede actuar como entrada de paquetes, para los paquetes que se reciben en dicha interfaz, y como interfaz de salida, para los paquetes que se envían a través de dicha interfaz.



Elementos del control de tráfico: Qdisc/Class/Filter

- Qdisc (Disciplina de cola): determina qué paquetes se reenviarán antes que otros. Las hay de 2 tipos:
  - Disciplina de colas **sin clases de tráfico**: ejemplo FIFO.

  - Disciplina de colas **con clases de tráfico**: ejemplo PRIO
    - **Class** (Clase de tráfico): especifica las diferentes categorías de tráfico. Una clase de tráfico está asociada a una disciplina de cola (qdisc).
    - **Filter** (Filtro de tráfico): identifica el tráfico que se corresponde con cada clase (class).



Configuración de una disciplina de cola a la entrada/salida de un router

- En una determinada interfaz se pueden definir:
  - **Disciplinas de cola de entrada**: para los paquetes que entran a través de dicha interfaz.

```
tc qdisc add dev <interfaz> ingress ...
```

  - **Disciplinas de cola de salida**: para los paquetes que salen a través de dicha interfaz.

```
tc qdisc add dev <interfaz> root ...
```

- Por defecto, si no se modifican las disciplinas de cola a la entrada y a la salida de un router, el router aplicará una variante de FIFO (`pfifo_fast`, FIFO mejorado).

Descriptor (*handle*) de una disciplina de cola

- Cuando se define una disciplina de cola se le asocia un descriptor (*handle*) para poder referenciarla.
- El *handle* está formado por 2 números separados por el caracter ":", es decir, `X:Y`
  - `X` : es el número mayor.
  - `Y` : es el número menor.

  Donde `X` identifica la disciplina de cola e `Y` hace referencia a un elemento que depende de esa disciplina de cola (flujos, clases).
- El *handle* de la disciplina de la cola de entrada (*ingress*) es `ffff:` lo que equivale a `ffff:0`.

```
tc qdisc add dev <interfaz> ingress handle ffff:
```

- Típicamente el *handle* de la disciplina de la cola de salida (*root*) es `1:` lo que equivale a `1:0`.

```
tc qdisc add dev <interfaz> root handle 1: ...
```

Borrado de una disciplina de cola

- El borrado de disciplinas de cola en la entrada y en la salida de un router se hace de la siguiente forma:
- Borrado de la disciplina de la cola de entrada:

```
tc qdisc del dev <interfaz> ingress
```

- Borrado de la disciplina de la cola de salida:

```
tc qdisc del dev <interfaz> root
```

Contenidos

Control de admisión para el tráfico de entrada (*policing*)

- **Objetivo**: El control de admisión para el tráfico de entrada se utiliza para garantizar que el router sólo procesa para un determinado flujo de paquetes de entrada un máximo de ancho de banda.
- El control de admisión se basa en el modelo TBF.
  - **Definición de la disciplina de cola de entrada**. Ejemplo, si los paquetes se reciben en la interfaz `eth0`:

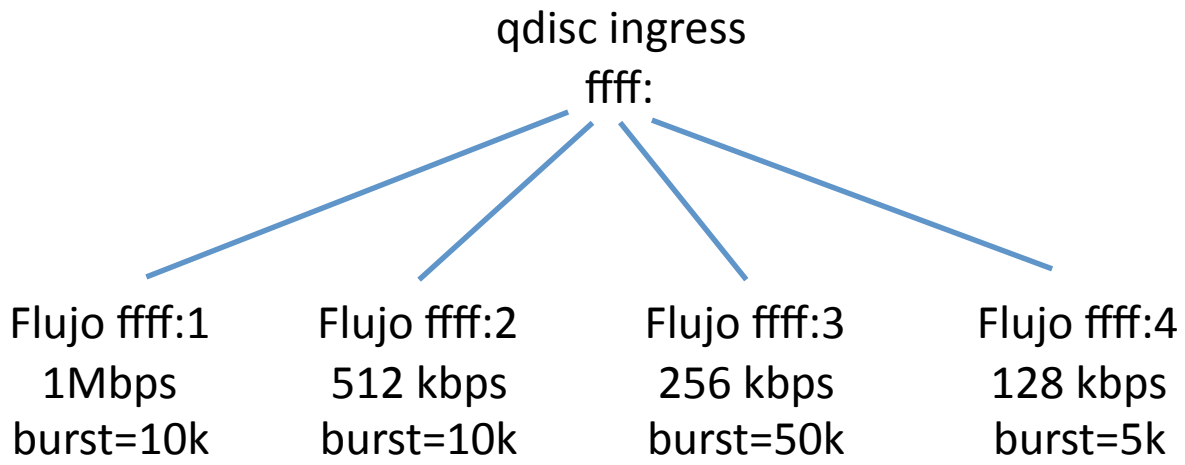
```
tc qdisc add dev eth0 ingress handle ffff:
```

  - **Definición de filtros con limitación de ancho de banda para cada flujo**: se especifican las características del tráfico en función de valores de cabecera IP y se indica el ancho de banda máximo y el tamaño de la cubeta TBF. El flujo queda identificado por un número menor :n.

```
tc filter add dev eth0 parent ffff: \
    protocol ip prio 4 u32 \
    match ip src 11.0.0.100/32 \
    police rate 1mbit burst 10k continue flowid :1
```

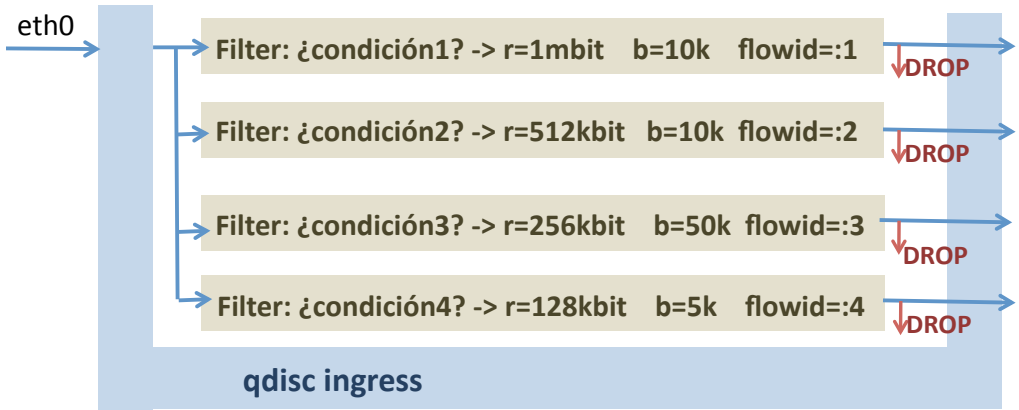
Ejemplo: control de admisión en el tráfico de entrada (I)

- Supongamos que se quiere separar el tráfico que se ha recibido en una determinada interfaz, en varios flujos.
- En la disciplina de cola de entrada (ffff:) queda almacenado todo el tráfico recibido en dicha interfaz.
- Ahora es necesario aplicar filtros a ese tráfico para separarlo en *i* flujos (cada flujo queda identificado por `ffff:i`) a los que asignaremos el ancho de banda que deseemos.



Ejemplo: control de admisión en el tráfico de entrada (II)

- Se define qdisc para la entrada: ingress
- Se definen filtros para clasificar el tráfico: los paquetes que cumplen una condición se les aplica un TBF y quedan clasificados dentro de un determinado flujo (*flowid*):
  - **Condición**: utilizaremos el filtro u32 que comprueba campos de la cabecera IP de los paquetes.
  - **Perfil de tráfico**: se configura con *rate* y *burst*.
  - **Flujo**: el tráfico que esté dentro de la especificación queda clasificado en un `flowid`.
  - **Acción**: el tráfico que supere la especificación puede reclasificarse pasando al siguiente filtro (*continue*) o descartarse (*drop*).
- Los filtros se aplican siguiendo el orden dado por el parámetro prioridad: primero los filtros cuyo valor de prioridad sea menor (números de prioridad bajos =>mayor prioridad).
- Si un paquete cumple la condición del filtro, se le aplica su clasificación o su acción asociada. Si no, se pasa al siguiente filtro por orden de prioridad.



Ejemplo: control de admisión en el tráfico de entrada (III)

- Ejemplo:

```
tc qdisc add dev eth0 ingress handle ffff:

tc filter add dev eth0 parent ffff: \
    protocol ip prio 4 u32 \
    match ip src 11.0.0.100/32 \
    police rate 1mbit burst 10k continue flowid :1

tc filter add dev eth0 parent ffff: \
    protocol ip prio 5 u32 \
    match ip src 11.0.0.100/32 \
    police rate 512kbit burst 10k continue flowid :2

tc filter add dev eth0 parent ffff: \
    protocol ip prio 6 u32 \
    match ip src 11.0.0.100/32 \
    police rate 256kbit burst 50k drop flowid :3

tc filter add dev eth0 parent ffff: \
    protocol ip prio 6 u32 \
    match ip src 0.0.0.0/0 \
    police rate 128kbit burst 5k  drop flowid :4
```

Contenidos

Contenidos

PFIFO para el tráfico de salida

- FIFO con tamaño de cola dado por el valor `limit` en número de paquetes:

```
tc qdisc add dev eth0 root pfifo limit 5
```

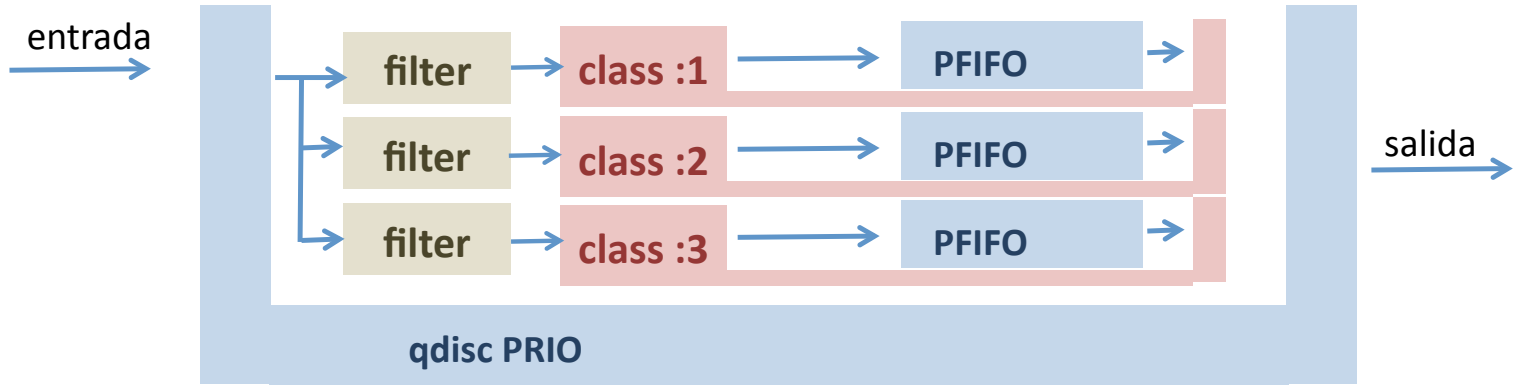
- Es una disciplina de colas muy sencilla que requiere poco tiempo de computación.
- Si hay congestión, FIFO perjudica a TCP debido a que una pérdida de un paquete en TCP activa los mecanismos de recuperación de pérdidas (si la ventana está llena, no se puede enviar más). Una pérdida de un paquete UDP no tiene ningún impacto en la tasa de envío para esta comunicación.
- Una ráfaga de uno de los flujos que atraviese una cola FIFO puede llenar la cola y perjudicar al resto de los flujos.

Contenidos

PRIO para el tráfico de salida (I)

- PRIO es una disciplina de cola de prioridades que realiza algunas acciones automáticamente. El siguiente comando crea la disciplina de cola 1:0 y automáticamente crea 3 clases 1:1, 1:2 y 1:3 de tipo PFIFO.

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



- Si se desea crear un número diferente de prioridades se puede usar el parámetro: `bands <núm_prio>`.

PRIO para el tráfico de salida (II)

- Es necesario especificar los filtros para clasificar el tráfico dentro de las clases 1:1, 1:2 y 1:3 del ejemplo. Esta clasificación puede hacerse a través del filtro u32 utilizando la dirección IP origen:

```
tc filter add dev eth0 parent 1:0 prio 1 protocol ip u32 \
          match ip src 11.0.0.1/32 flowid 1:1
tc filter add dev eth0 parent 1:0 prio 2 protocol ip u32 \
          match ip src 11.0.0.2/32 flowid 1:2
tc filter add dev eth0 parent 1:0 prio 3 protocol ip u32 \
          match ip src 11.0.0.3/32 flowid 1:3
```

- La disciplina de colas de prioridad puede provocar retrasos y pérdidas de paquetes en la clase menos prioritaria 1:3 si hay mucho tráfico en 1:1 y 1:2.
- Para solventar que TCP se vea perjudicado con PFIFO, se podría clasificar tráfico TCP en 1:1 y el UDP en 1:2. Sin embargo, si hubiera muchos datos para TCP se vería perjudicado UDP ya que mientras hubiera tráfico en 1:1 no se cursaría el resto de tráfico.

Contenidos

Token Bucket Filter (TBF) para el tráfico de salida

- Disciplina de cola sin clases, trata a todos los paquetes por igual.
- TBF limita la velocidad del tráfico de salida de una determinada interfaz de un router a través de los parámetros:
  - ancho de banda: velocidad de generación tokens
  - tamaño de la cubeta: para almacenar tokens que permitirán enviar ráfagas que superen el ancho de banda.
  - latencia: tiempo máximo de espera de un paquete por un token.
- Ejemplo: Aplicar TBF en la interfaz de salida eth1 de un router. Ancho de banda 7Mbps , tamaño de cubeta 10k (en bytes), latencia 50 ms:

```
tc qdisc add dev eth1 root handle 1: tbf  rate 7Mbit \
            burst 10k latency 50ms
```

Contenidos

TBF y PRIO

- TBF limita la velocidad del tráfico de salida de una determinada interfaz para todos los flujos que lo atraviesan.
- Se puede definir una disciplina de cola PRIO hija de TBF para que el tráfico además de estar limitado por TBF los paquetes se cursen atendiendo a las prioridades definidas en PRIO:

```
tc qdisc add dev eth1 root handle 1: tbf rate 7Mbit \
            burst 10k latency 50 ms

tc qdisc add dev eth1 parent 1:0 handle 10:0 prio

tc filter add dev eth1 parent 10:0 prio 1 protocol ip u32 \
     match ip src 101.0.0.10/32 flowid 10:1
tc filter add dev eth1 parent 10:0 prio 2 protocol ip u32 \
     match ip src 102.0.0.20/32 flowid 10:2
tc filter add dev eth1 parent 10:0 prio 3 protocol ip u32 \
     match ip src 103.0.0.30/32 flowid 10:3
```

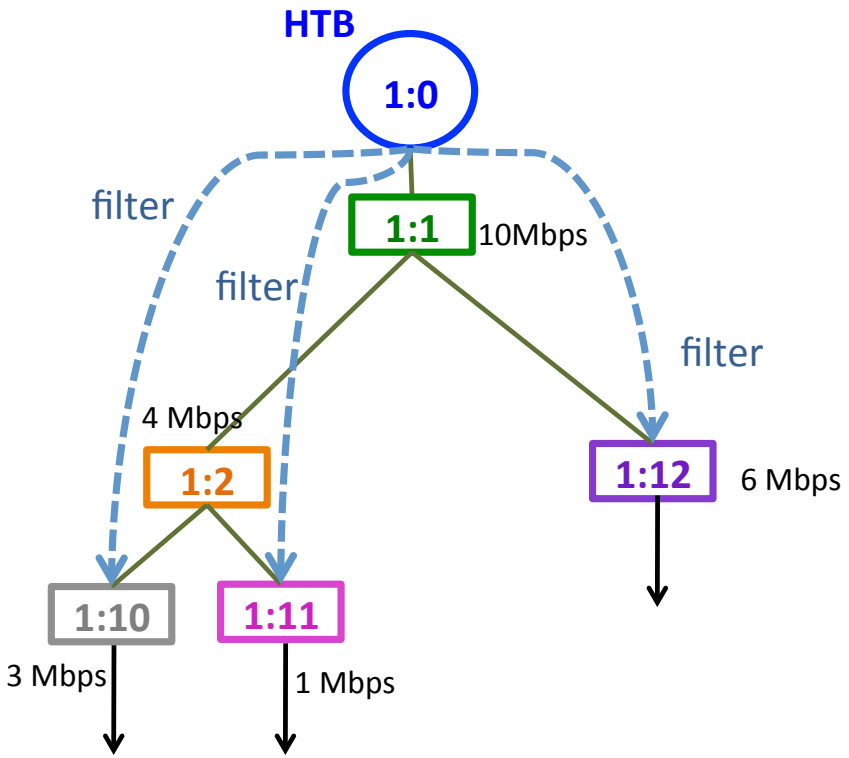
Contenidos

Hierarchical Token Bucket (HTB) para el tráfico de salida

- TBF se utiliza cuando se desea que todo el tráfico que sale por una determinada interfaz tenga unas características determinadas en cuanto a ancho de banda, tamaño de cubeta y latencia.
- HTB es necesario cuando se quiere clasificar el tráfico que sale por una determinada interfaz en varias clases, cada uno de ellas con unas características diferentes de ancho de banda.
- Si alguna clase no utiliza todo el ancho de banda que se le ha definido, dependiendo de la configuración, se podría utilizar dicho ancho de banda para algún otra clase que lo necesite.

Ejemplo de Hierarchical Token Bucket (HTB) para el tráfico de salida

- Definición de HTB con ancho de banda máximo de salida de 10Mbps de la interfaz eth0. Se quiere repartir este ancho de banda:
  - usuarioA (4Mbps): lo usará para tráfico HTTP (3Mbps) y tráfico de correo (1Mbps)
  - el usuarioB (6Mbps)



Ejemplo de Hierarchical Token Bucket (HTB) para el tráfico de salida

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

tc class add dev eth0 parent 1:0 classid 1:1 htb rate 10Mbit

tc class add dev eth0 parent 1:1 classid 1:2 htb rate 4Mbit ceil 10Mbit
tc class add dev eth0 parent 1:2 classid 1:10 htb rate 3Mbit ceil 10Mbit
tc class add dev eth0 parent 1:2 classid 1:11 htb rate 1Mbit ceil 10Mbit
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 6Mbit ceil 10Mbit

tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 \
  match ip src 1.1.1.1 \
  match ip dport 80 0xffff \
  flowid 1:10

tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 \
  match ip src 1.1.1.1 \
  match ip dport 25 0xffff \
  flowid 1:11

tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 \
  match ip src 2.2.2.2 \
  flowid 1:12
```

Unidades para tc

- Ancho de banda

kbps	Kilobytes per second
mbps	Megabytes per second
kbit	Kilobits per second
mbit	Megabits per second
bps	Bytes per second

- Cantidad de datos

k	Kilobytes
mb o m	Megabytes
kbit	Kilobits
mbit	Megabits
b	Bytes

- Tiempo

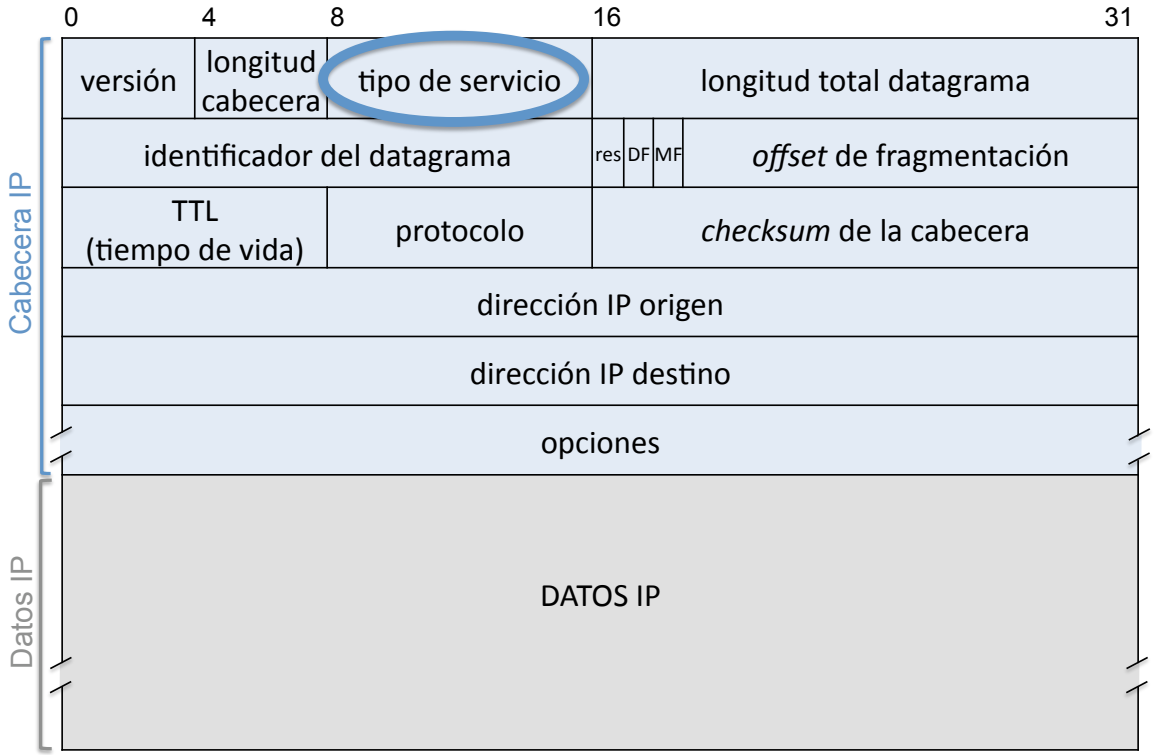
s, sec o secs	Segundos
ms, msec o msecs	Milisegundos
us, usec, usecs	Microsegundos

Contenidos

Contenidos

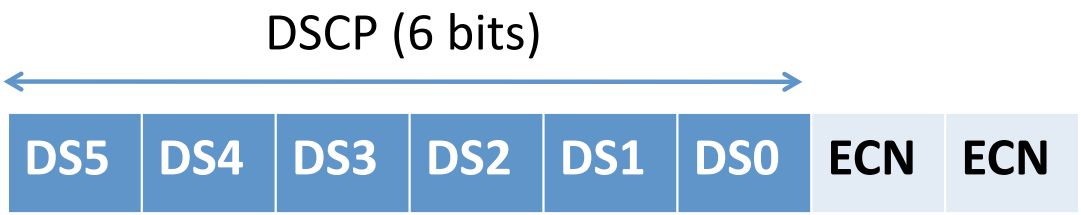
Marcado del tráfico DiffServ en la cabecera IPv4

- Los paquetes se marcan en el campo de 8 bits Type of Service (ToS) de IPv4



Marcado del tráfico DiffServ: campo DSCP

- Se usan 6 bits para identificar *Differentiated Service Code Point (DSCP)* que determinan el comportamiento por salto (PHB, Per-Hop Behavior) que recibirá el paquete en los *routers* de la red DiffServ.
- Quedan los 2 bits menos significativos del campo ToS que no se usan para DiffServ, sino para la notificación de congestión (Explicit Congestion Notification, ECN). ECN es utilizado conjuntamente por los extremos de una conexión TCP y los routers intermedios que utilizan en sus colas *Active Queue Management* (monitorizar el llenado de sus buffers santes de que se produzca la congestión).



Comportamiento por salto (PHB)

- Cada clase de tráfico (DSCP) está asociado a un comportamiento por salto (*PHB, Per Hop Behavior*).
- El PHB determina el tipo de tratamiento que se le va a dar al paquete en el reenvío:
  - Reserva de recursos: buffer y ancho de banda.
  - Características del tráfico: retardo y pérdidas.
- DiffServ no especifica las características concretas del tráfico en cada una de las clases, sólo proporciona la clasificación con diferentes códigos DSCP.

PHBs recomendados

- Aunque potencialmente pueden definirse 64 clases de tráfico diferente, y puede establecerse un PHB para cada una de ellas, la RFC recomienda al menos los siguientes PHB:
  - **EF** (Expedited Forwarding): DSCP 46=101110_B
    - Bajas pérdidas, baja latencia, bajo jitter
  - **VA** (Voice Admit): DSCP 44=101100_B
    - Similar a EF, pero puede incluir un mecanismo de control de admisión
  - **AF** (Assured Forwarding):
    - Se proporciona cierta garantía de entrega mientras el tráfico no exceda una tasa acordada.
    - Define 4 clases difentes: AF1-AF4. Dentro de cada clase se crean 3 PHBs, cada uno con diferente probabilidad de pérdida de los paquetes. La combinación de las 4 clases con las 3 probabilidades de pérdida dan 12 PHBs: AF11, AF12, AF13, AF21, AF22, AF23, AF31, AF32, AF33, en los que se cumple que la probabilidad de pérdida de AFX1 ¡ AFX2 ¡ AFX3 para cada una de las cuatro clases.
  - **CS** (Class Selector): usa los 3 primeros bits DSCP=XXX000 para definir prioridades (compatible con el antiguo ToS).
    - Menor prioridad = 000000 a mayor prioridad = 111000
  - **DF** (Default Forwarding): DSCP 0=000000_B
    - IP Best Effort (compatible con tráfico que no es DiffServ)

Marcas en los paquetes: Valores DS y DSCP más comunes

- El valor DS incluye los 8 bits que viajan en el campo TOS de la cabecera IP. Estos 8 bits son incluyen DSCP (6 bits) + ECN (2 bits).

		DSCP (6bits)	DS (8 bits)	Probabilidad de descarte de tráfico
Clase DF		000 000 = 0x00 = 0	0000 0000 = 0x00	
Clase AF1	AF11	**001** 010 = 0x0a = 10	0010 1000 = 0x28	Baja
	AF12	**001** 100 = 0x0c = 12	0011 0000 = 0x30	Media
	AF13	**001** 110 = 0x0e = 14	0011 1000 = 0x38	Alta
Clase AF2	AF21	**010** 010 = 0x12 = 18	0100 1000 = 0x48	Baja
	AF22	**010** 100 = 0x14 = 20	0101 0000 = 0x50	Media
	AF23	**010** 110 = 0x16 = 22	0101 1000 = 0x58	Alta
Clase AF3	AF31	**011** 010 = 0x1a = 26	0110 1000 = 0x68	Baja
	AF32	**011** 100 = 0x1c = 28	0111 0000 = 0x70	Media
	AF33	**011** 110 = 0x1e = 30	0111 1000 = 0x78	Alta
Clase AF4	AF41	**100** 010 = 0x22 = 34	1000 1000 = 0x88	Baja
	AF42	**100** 100 = 0x24 = 36	1001 0000 = 0x90	Media
	AF43	**100** 110 = 0x26 = 38	1001 1000 = 0x98	Alta
Clase EF		101 110 = 0x2e = 46	1011 1000 = 0xb8	

Contenidos

DSMARK: qdisc

- DSMARK es la qdisc que se utiliza para la clasificación de paquetes según la arquitectura de DiffServ y el marcado del campo DS.
- Marcado del campo DS:
  - **Routers frontera de DiffServ**: identifican clases en los paquetes atendiendo a valores de su cabecera IP y marcan adecuadamente el campo DS.
- Clasificación del tráfico según el campo DS:
  - **Routers del núcleo de DiffServ**: clasifican los paquetes atendiendo al contenido de DS que traen para realizar su tratamiento según diferentes disciplinas de cola.

DSMARK: qdisc y sus clases

- Una qdisc DSMARK crea automáticamente un conjunto de clases para asignar a cada una de ellas un valor diferente en el campo DSCP.
- Al crear la qdisc DSMARK es necesario especificar la cantidad máxima de valores DSCP que se van a usar para clasificar los paquetes. Este valor debe ser una potencia de 2.
  - Por ejemplo, la siguiente qdisc con número de clases igual a $2^3 = 8$ podrá definir como máximo 7 valores DSCP diferentes, uno para cada una de sus clases:

```
tc qdisc add dev eth1 root handle 1:0 dsmark indices  8
```

- Una vez creada una qdisc DSMARK se habrán creado automáticamente sus clases asociadas. Cada clase tiene un descriptor X:Y, donde:
  - X es el valor del número mayor de la clase qdisc a la que pertenece
  - Y es el número menor que distingue a cada una de las clases
- Por ejemplo, para definir como máximo 7 clases de tráfico dentro de una qdisc 1:0 se usarán los siguientes descriptores: 1:1, 1:2, 1:3, 1:4, 1:5, 1:6, 1:7. El número mayor coincide en todas ellas, ya que todas pertenecen a la qdisc 1.

Contenidos

Router Frontera: marcado de paquetes

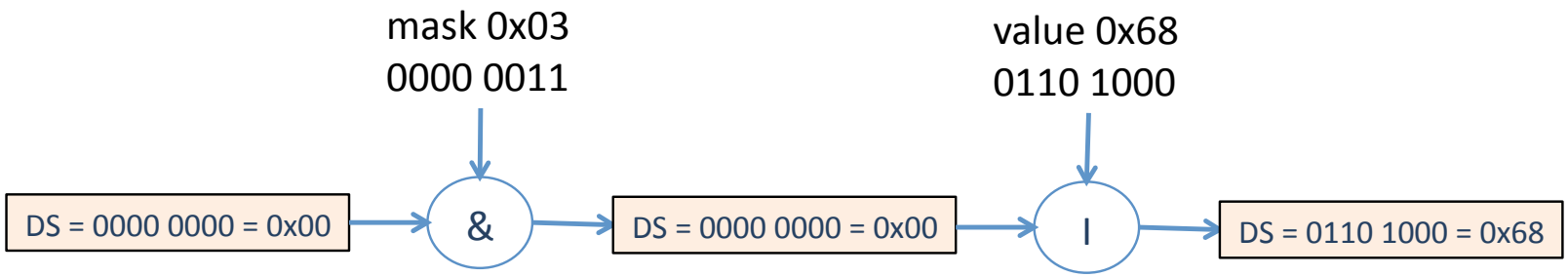
- Para marcar el tráfico se usa la disciplina DSMARK:

```
tc qdisc add dev eth1 root handle 1:0 dsmark indices  8
```

- Esta disciplina ha creado 7 clases, se desea que los paquetes de cada clase queden marcados en el campo DS (6 bits más significativos). Para ello, se realizan las 2 siguientes operaciones en orden:
  1. operación & (AND) de bits con una máscara para borrar los bits DSCP (6 bits) y mantener ECN (2 bits)
  2. operación | (OR) de bits con el valor DS que se desee asignar.

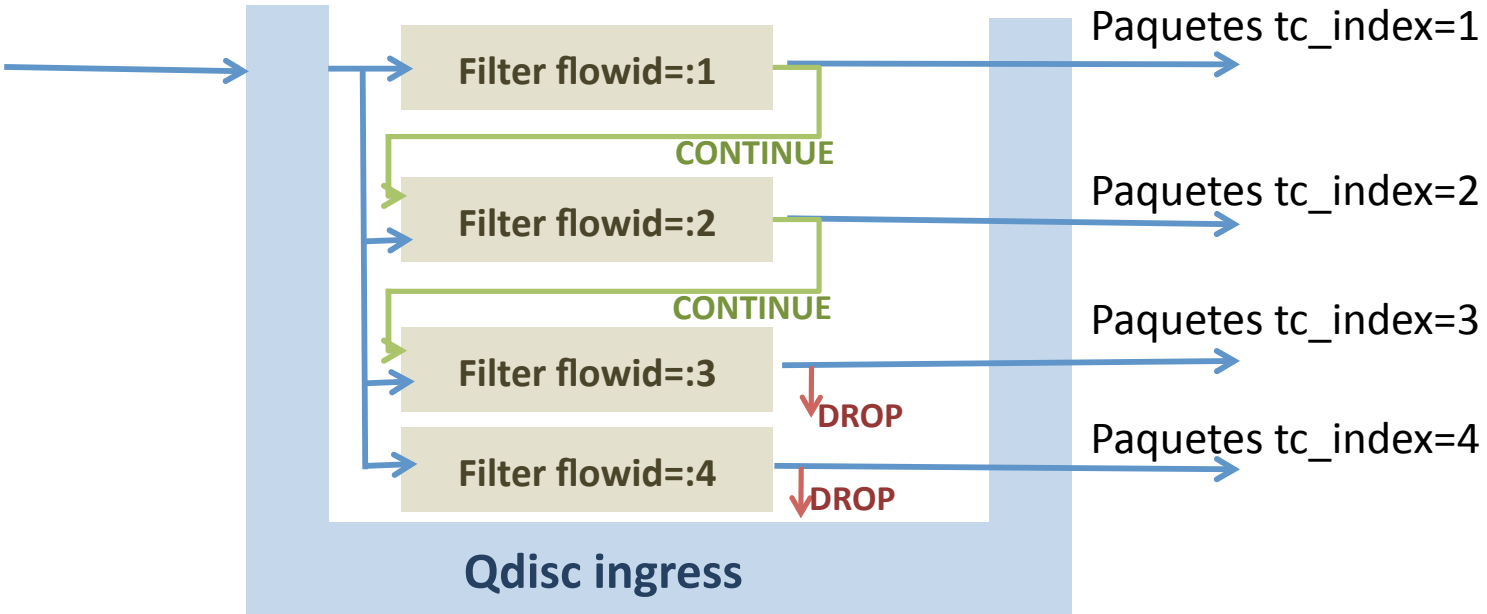
- Por ejemplo: para que la clase 1:3 se marque con DS=0x68 (AF31) conservando los bits ECN:

```
tc class change dev eth1 classid 1:3 dsmark mask 0x3 value 0x68
```



Campo tc_index

- Al recibirse un paquete IP en un router Linux, se almacena en un buffer.
- Cuando un paquete atraviesa la disciplina de cola de entrada `qdisc ingress`, el buffer que almacena el paquete IP dentro del kernel del router Linux tiene un **campo tc_index** en el que se almacena el identificador de flujo en el que fue clasificado dicho paquete cuando ingresó en la cola qdisc ingress:



Campo tc_index en la disciplina ingress

- Si un paquete queda clasificado en `flowid :2`, el campo `tc_index` almacenaría el valor 2.
- Esta información es la que utiliza posteriormente el filtro tcindex dentro de otras qdisc de dicho router para dar tratamiento diferenciado a los paquetes.

```
tc qdisc add dev eth0 ingress handle ffff:

tc filter add dev eth0 parent ffff: \
    protocol ip prio 4 u32 \
    match ip src 11.0.0.100/32 \
    police rate 1mbit burst 10k continue flowid :1

tc filter add dev eth0 parent ffff: \
    protocol ip prio 5 u32 \
    match ip src 11.0.0.100/32 \
    police rate 512kbit burst 10k continue flowid :2

tc filter add dev eth0 parent ffff: \
    protocol ip prio 6 u32 \
    match ip src 11.0.0.100/32 \
    police rate 256kbit burst 10k drop flowid :3

tc filter add dev eth0 parent ffff: \
    protocol ip prio 6 u32 \
    match ip src 0.0.0.0/0 \
    police rate 128kbit burst 10k  drop flowid :4
```

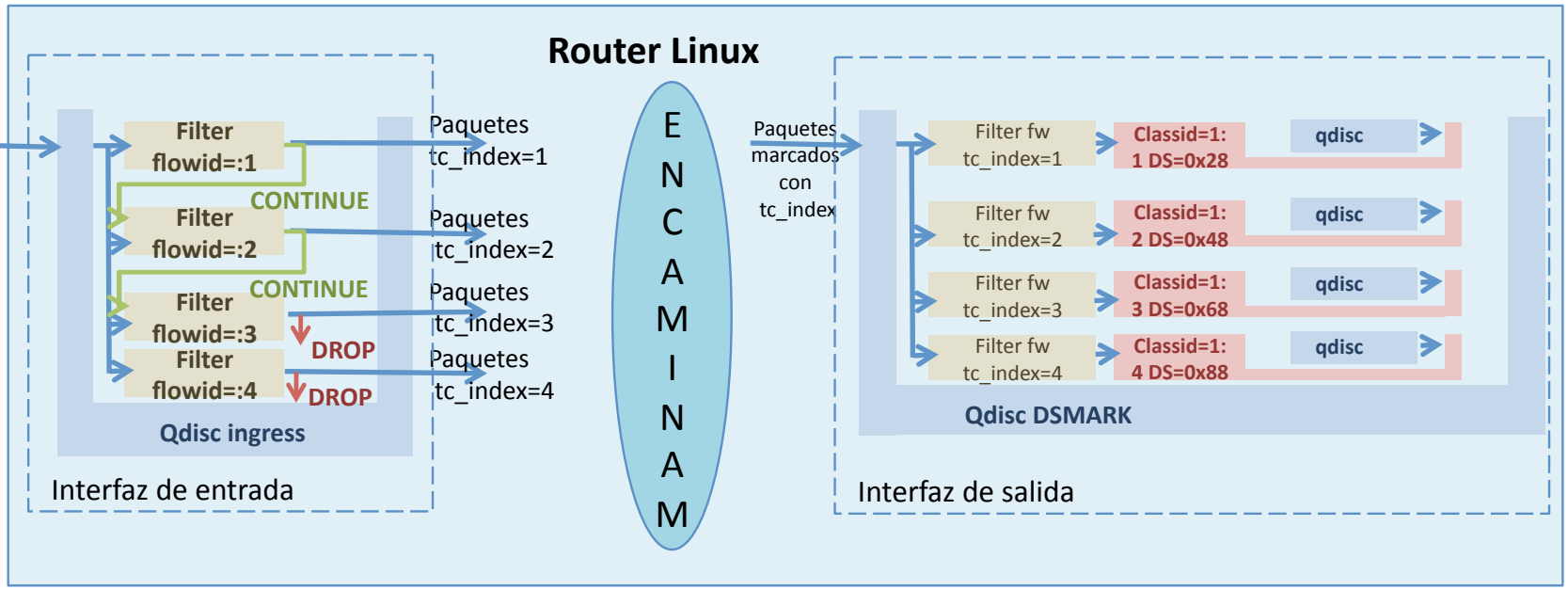
Filtro tcindex en el Router Frontera

- EL filtro `tcindex` utiliza el valor almacenado en `tc_index` del buffer que contiene un paquete IP para clasificarlo dentro de una clase en la disciplina de cola de salida, en este caso DSMARK.
- Ejemplo:
  - qdisc DSMARK con 3 diferentes valores de marcado

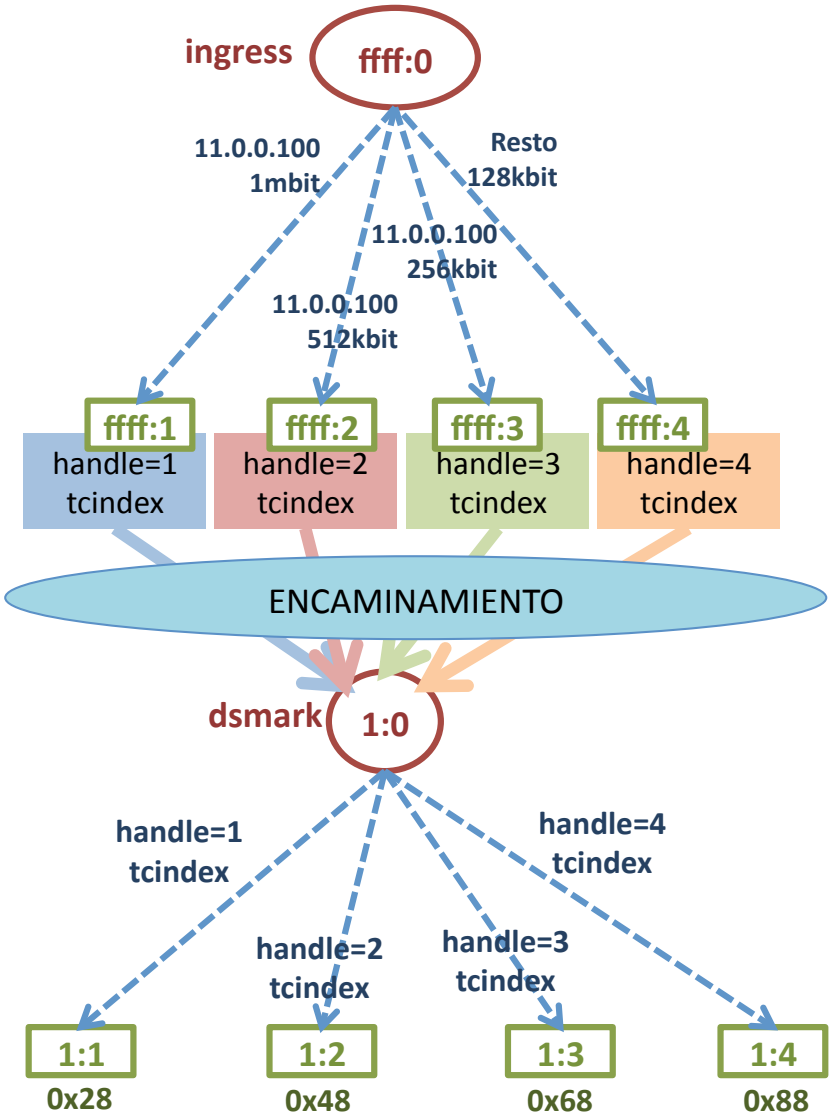
```
tc qdisc add dev eth1 root handle 1:0 dsmark indices 8

tc class change dev eth1 classid 1:1 dsmark mask 0x3 value 0x28
tc class change dev eth1 classid 1:2 dsmark mask 0x3 value 0x48
tc class change dev eth1 classid 1:3 dsmark mask 0x3 value 0x68
tc class change dev eth1 classid 1:4 dsmark mask 0x3 value 0x88

tc filter add dev eth1 parent 1:0 protocol ip prio 1 \
        handle 1 tcindex classid  1:1
tc filter add dev eth1 parent 1:0 protocol ip prio 1  \
        handle 2 tcindex classid  1:2
tc filter add dev eth1 parent 1:0 protocol ip prio 1  \
        handle 3 tcindex classid  1:3
tc filter add dev eth1 parent 1:0 protocol ip prio 1  \
        handle 4 tcindex classid  1:4
```

Camino de un paquete en un Router Frontera

- Los paquetes entran en el router y atraviesan la cola ingress, donde se rellena el campo `tc_index`.
- Dentro de la cola DSMARK en la interfaz de salida, se comprueba el valor de `tc_index` y se clasifica el tráfico en las clases de DSMARK, que marcarán los paquetes con un determinado valor de DS.



Filtro tcindex: Router Frontera (VI)



Contenidos

Campo tc_index

- En el **router del núcleo**, el paquete llega con el valor DS establecido.
- Si el router no tiene configurada una disciplina de cola ingress, inicialmente no se rellena el campo `tc_index` del buffer del kernel del Linux que contiene el paquete.
- La propia qdisc DSMARK puede copiar el campo DS que lleva la cabecera IP del paquete en el campo `tc_index` de la estructura de datos del kernel del Linux que almacena el paquete utilizando el siguiente parámetro: set_tc_index

```
tc qdisc add dev eth1 root handle 1:0 dsmark indices 8 set_tc_index
```

Disciplinas de cola encadenadas DSMARK+HTB

- Como se copia el campo DS entero, el filtro deberá utilizar sólo los 6 bits más significativos del campo DS, es decir, el DSCP, para clasificar los paquetes. Es necesario extraer esos 6 bits (operación & con mask 1111 1100 = 0xfc y desplazamiento a la derecha 2 bits) del campo DSCP:

```
tc filter add dev eth1 parent 1:0 protocol ip prio 1  \
     tcindex mask 0xfc shift 2
```

- A continuación se puede usar otra disciplina de cola encadenada con DSMARK para realizar la planificación de paquetes que se desee, por ejemplo HTB:

```
tc qdisc add dev eth1 parent 1:0  handle 2:0 htb

tc class add dev eth1 parent 2:0 classid 2:1 htb rate 1Mbit
tc class add dev eth1 parent 2:1 classid 2:10 htb rate 800kbit ceil 1Mbit

# Tráfico AF11=> DS=0x28 =00101000 =>  DSCP=001010=0x0a
tc filter add dev eth1 parent 2:0 protocol ip prio 1 \
     handle 0x0a tcindex classid 2:10
```

Router del Núcleo

- Ejemplo completo:

```
tc qdisc add dev eth1 root handle 1:0 dsmark indices 8 set_tc_index

tc filter add dev eth1 parent 1:0 protocol ip prio 1  \
      tcindex mask 0xfc shift 2

tc qdisc add dev eth1 parent 1:0 handle 2:0 htb

tc class add dev eth1 parent 2:0 classid 2:1 htb rate 1Mbit
tc class add dev eth1 parent 2:1 classid 2:10 htb \
     rate 800kbit ceil 1Mbit

# Tráfico AF11=> DS=0x28 =00101000 =>  DSCP=001010=0x0a
tc filter add dev eth1 parent 2:0 protocol ip prio 1 \
      handle 0x0a tcindex classid 2:10
```

Contenidos

Iperf: Generador de tráfico

- Iperf es una aplicación que permite generar tráfico TCP o UDP y medir el ancho de banda que se está utilizando entre dos máquinas finales.
- Iperf funciona en modo cliente/servidor.
- Utilizaremos Iperf para generar tráfico UDP desde el cliente al servidor durante 10 segundos a una determinada velocidad y medir el ancho de banda que en realidad se ha utilizado.
- Arrancaremos Iperf de la siguiente forma:
  - Servidor

```
iperf -u -s
```

  - Cliente

```
iperf -u -c <dirIPServidor> -b <anchoDeBanda>
```

    Donde `<anchoDeBanda>` es un número `n` en bps. Se puede usar nK (para enviar n kilobit por segundo) o nM (para enviar n megabit por segundo).

Resultado de la ejecución de Iperf

- Resultado de ejecución de Iperf en 12.0.0.30, en modo servidor:

```
iperf -s -u
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
------------------------------------------------------------
[3] local 12.0.0.30 port 5001 connected with 11.0.0.10 port 32768
[3] 0.0- 9.7 sec 25.2 MBytes 21.8 Mbits/sec 0.289 ms 4940/22916 (22%)
[3] 0.0- 9.7 sec 1 datagrams received out-of-order
```

- Resultado de ejecución de Iperf en 11.0.0.10, en modo cliente:

```
iperf -u -c 12.0.0.30 -b 100M
------------------------------------------------------------
Client connecting to 12.0.0.30, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size: 108 KByte (default)
------------------------------------------------------------
[3] local 11.0.0.10 port 32768 connected with 12.0.0.30 port 5001
[3] 0.0- 10.0 sec 32.1 MBytes 26.9 Mbits/sec
[3] Sent 22917 datagrams
[3] Server Report:
[3] 0.0- 9.7 sec 25.2 MBytes 21.8 Mbits/sec 0.289 ms 4940/22916 (22%)
[3] 0.0- 9.7 sec 1 datagrams received out-of-order
```

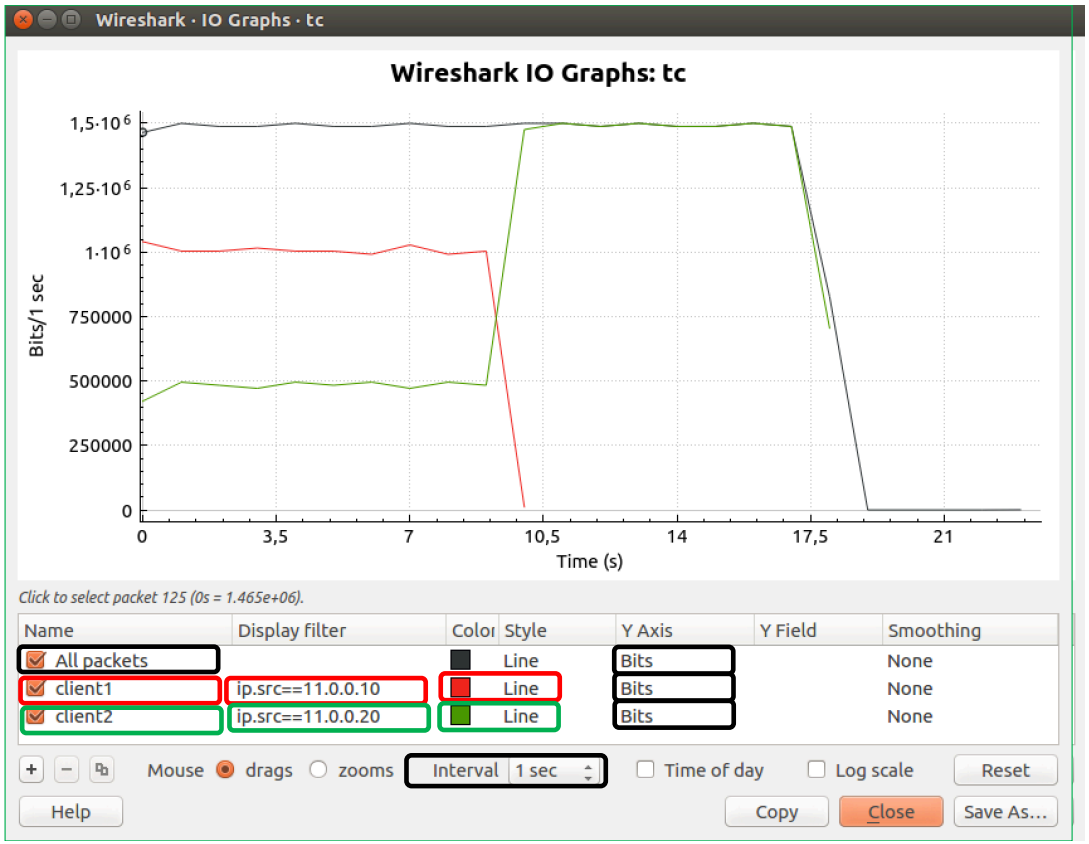
Contenidos

Gráficas de ancho de banda (I)

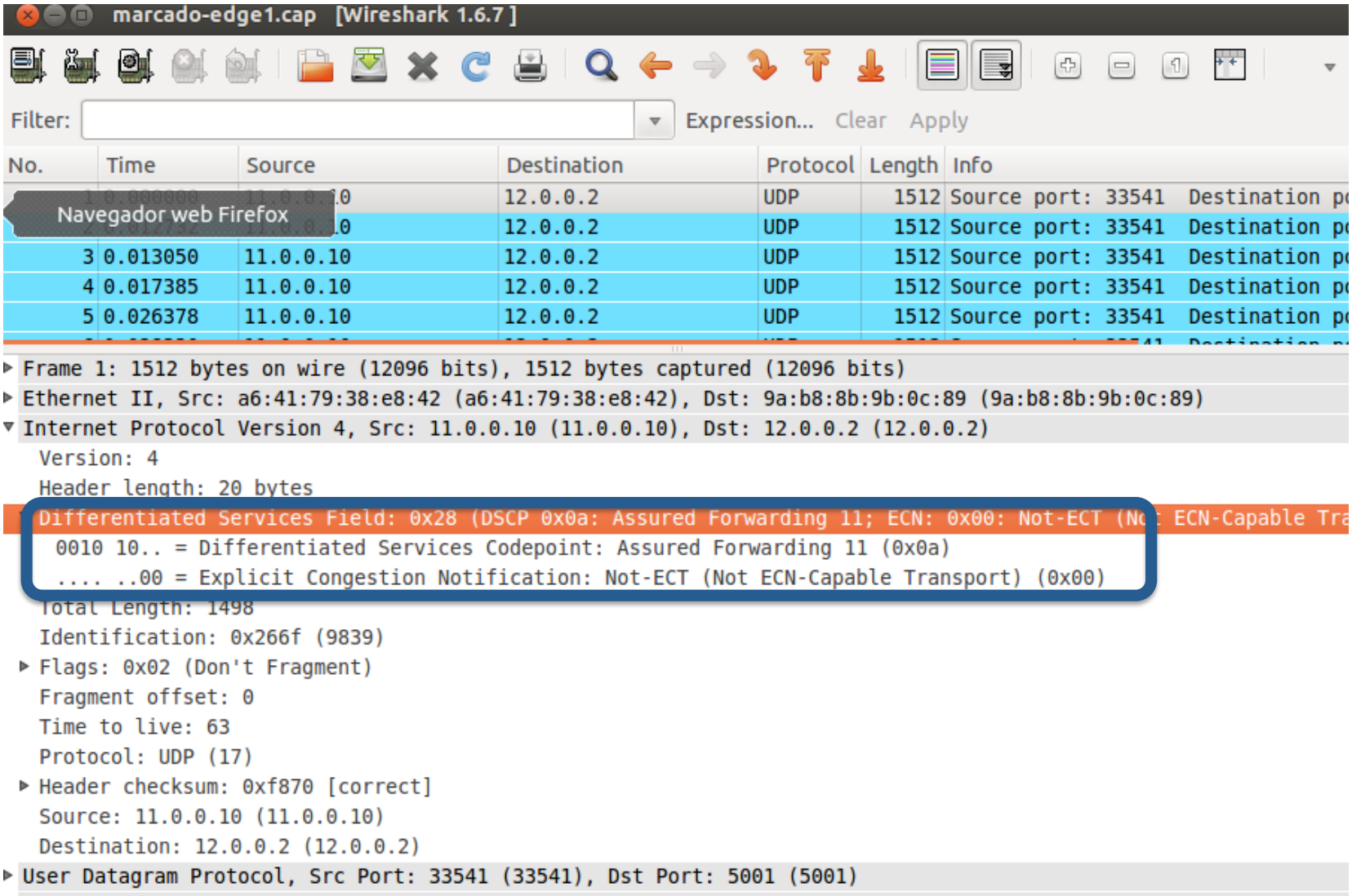
- Seleccionar en el menú de Wireshark `Statistics` $->$ `IO Graphs`.
- En el panel inferior, añadir filtros según los criterios que se deseen. Marcar con un tic a la izquierda del filtro para visualizar los paquetes filtrados en la gráfica.
- Para distinguir el tráfico de varias fuentes la condición de filtrado en `Display filter` puede incluir la comprobación de la dirección IP y puerto TCP/UDP. Ejemplo: `ip.src==11.0.0.1 and udp.dstport==5001`
- Si no se escribe nada en la condición de filtrado, se muestra la gráfica de todo el tráfico incluido en la captura.

Gráficas de ancho de banda (II)

- Es importante seleccionar adecuadamente las unidades que muestra la gráfica. Para las pruebas que vamos a realizar en las prácticas se recomienda:
  - Color: Uno diferente para cada filtro
  - Style: Line
  - Y Axis: Bits
  - Interval (común para todos los filtros): 1 sec



Campo DS en una captura de Wireshark



Gráficas de ancho de banda en diferentes clases DiffServ (I)

- Seleccionar en el menú de Wireshark `Statistics` $->$ `IO Graphs`.
- En el panel inferior, añadir filtros según los criterios que se deseen. Marcar con un tic a la izquierda del filtro para visualizar los paquetes filtrados en la gráfica.
- Para filtrar una clase de tráfico puede usarse como condificón de filtrado alguna de las siguientes (cualquier de ellas serviría para filtrar el tráfico EF):
  - `ip.dsfield==0xb8`
  - `ip.dsfield.dscp==46`
  - `ip.dsfield.dscp==0x2e`
- Para distinguir el tráfico de varias fuentes la condición de filtrado puede completarse con la comprobación de la dirección IP. Ejemplo: `ip.dsfield==0x28 and ip.src==11.0.0.1`

Gráficas de ancho de banda en diferentes clases DiffServ