EL COMANDO TC

Planificación e Integración de Redes y Servicios

1 .INTRODUCCIÓN

Los enlaces y conexiones entre redes son hoy día muy veloces, es normal encontrar enlaces multipropósito, es decir, enlaces con conexiones que sirven de soporte para varios tipos de servicio, como: internet, correo corporativo, videoconferencia, canales de voz, consulta a servidores externos, etc.

Aunque estas redes estén configuradas para trabajar a 100Mbps o 10Mbps, no siempre (por no decir nunca) es posible alcanzar estas velocidades de transmisión. Esto es debido a la gran cantidad de colisiones que presenta *ethernet*, sin duda la topología más utilizada en la actualidad. Debido a esto, es muy recomendable tener bien definidos qué servicios son los fundamentales y cuánto ancho de banda requieren.

Pero no sólo es importante definir las prioridades en cuando a los servicios, también es importante definir las prioridades de las diferentes subredes que puedan existir por ejemplo en una misma empresa, por ejemplo, no se le puede asignar el mismo ancho de banda a áreas como desarrollo o gerencia y a áreas como atención al cliente o archivo. Es decir, no puede permitirse que el gerente de la empresa participe en una videoconferencia “entrecortada” porque alguien del archivo esté descargándose canciones de internet. Pensemos que esta misma persona, después de haber descargado estas canciones, desea enviárselas, por correo corporativo, a sus compañeros de empresa; la gente de desarrollo no podría acceder a sus bases de datos debido al uso excesivo del ancho de banda consumido por el tráfico de los correos. Para que todo lo anterior no ocurra, será necesaria una buena administración de los recursos.

El comando ***tc*** va a ayudar a administrar estos recursos.

2 .QUÉ ES “*tc*”

***tc* es un comando** de los sistemas *Linux* **que se utiliza para manipular y mostrar la configuración del control de tráfico** de una red.

2.1 Elementos del control de tráfico

El control de tráfico de *tc* está formado por los siguientes elementos:

- ***SHAPING (modelado)***

El modelado es un mecanismo por el cual la emisión de paquetes se retrasa, colocándolos en una cola de salida, **para cumplir con la tasa de salida deseada** (normalmente medida en paquetes por segundo o bits por segundo). El modelado puede consistir en algo más que reducir el ancho de banda disponible, también se utiliza para suavizar los picos de tráfico para un mejor comportamiento de la red.

El modelado de tráfico propone conceptos de clasificación, colas, imposición de políticas, administración de congestión, calidad de servicio y regulación.

La capacidad de controlar la latencia de los paquetes permite optimizar o garantizar el rendimiento y conseguir una baja latencia y/o un ancho de banda determinado.

- ***SCHEDULING (planificación)***

La planificación es el mecanismo por el cual los paquetes se ordenan (o reordenan) entre la entrada y la salida de una cola determinada. El programador más común es el FIFO (primero en entrar primero en salir), aunque también existen otros.

Mediante la planificación de la transmisión de paquetes es posible mejorar la interactividad para el tráfico que la necesita sin dejar de garantizar el ancho de banda para otras transferencias. La reordenación es también llamada priorización y también ocurre sólo para el tráfico de salida.

- ***POLICING (políticas)***

Mientras el modelado se ocupa de la transmisión del tráfico saliente, las políticas hacen lo propio con el tráfico entrante.

Una política es un mecanismo que mide y limita el tráfico de una cola de entrada particular. Si un paquete está a punto de entrar en la cola y este no supera la tasa establecida, se permitirá el encolado. Si el paquete que está a punto de entrar supera la tasa, se tomará otra acción. Aunque la política utiliza un mecanismo interno de cubos de símbolos, no tiene la capacidad de retrasar un paquete como si lo hacía el mecanismo de modelado.

- ***DROPPING (reducción)***

La reducción de paquetes es un mecanismo por el cual se descartan todos los paquetes de un tipo determinado. El mecanismo de reducción es recomendable si el tráfico excede de un ancho de banda determinado.

Este mecanismo es aplicable tanto para el tráfico entrante como para el saliente.

2.2 Objetos que intervienen en el control de tráfico

El procesamiento del tráfico está controlado por tres tipos de objetos: las *qdiscs*, las clases y los filtros.

- ***QDISCS***:
  Es la abreviatura de "queueing discipline" (disciplina de colas) y es fundamental para comprender el control del tráfico. **Cuando el kernel necesita enviar un paquete a una interfaz, este es encolado en la *qdisc* configurada para esa interfaz. Inmediatamente después, el kernel trata de obtener de la *qdisc* tantos paquetes como le sea posible, para dárselos después al driver del adaptador de red.**

  Una *qdisc* simple es una "*pfifo*" que no realiza ningún procesamiento y se limita a implementar una cola FIFO. Esta cola también se encarga de almacenar el tráfico cuando, en un momento dado, la interfaz de red no puede manejarlo.

- **CLASES:**
  Algunas *qdisc* pueden contener clases, que contienen a su vez más *qdisc*. El tráfico puede entonces ser encolado en alguna de esas *qdisc* internas. Cuando el kernel intenta desencolar un paquete de una *qdisc* con clases, este puede provenir de cualquiera de esas clases.

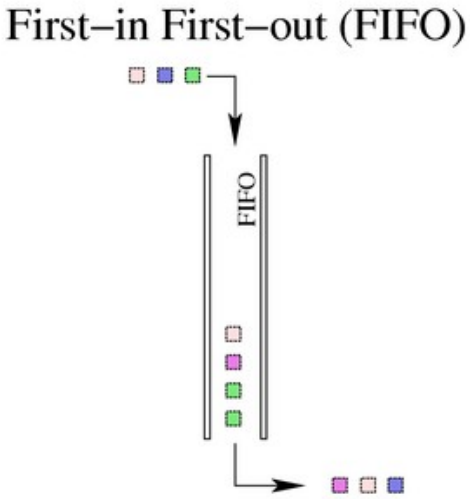
  Las *qdisc* con clases pueden utilizarse, por ejemplo, para priorizar cierto tipo de tráfico mediante el desencolado de unas clases antes que de otras.

- **FILTROS:**
  Un filtro es utilizado por una *qdisc* con clase para determinar en qué clase será encolado un paquete. Cuando el tráfico llega de una clase con subclases, este necesitará ser clasificado. El uso de filtros es uno de los métodos que pueden ser empleados para ello. Cuando un paquete llega a la *qdisc*, todos los filtros asociados a su clase son llamados, hasta que uno de ellos regresa con un veredicto. Si no hay veredicto, será necesario utilizar otros criterios.

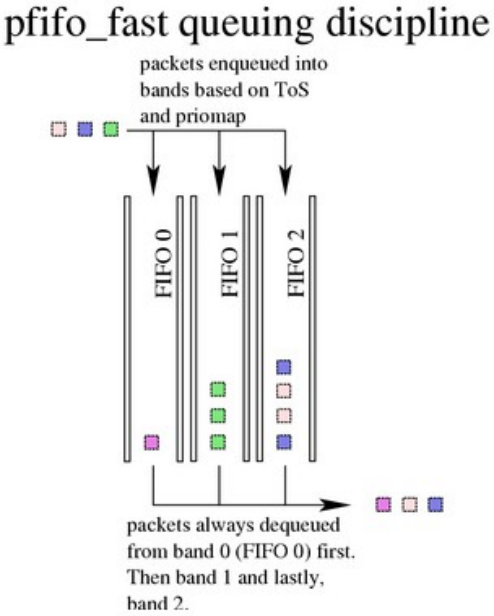
  Es importante tener en cuenta que los filtros residen dentro de las *qdisc* y que no son conscientes de lo que sucede.

Ahora vamos a ver las distintas implementaciones disponibles para las *qdiscs:*

- ***qdiscs* sin clases:**
  Las *qdisc sin clases pueden ser:*
  - ***fifo*** → Son las *qdisc* más simples. Tienen un comportamiento FIFO puro. Pueden limitarse por número de paquetes o por número de bytes.

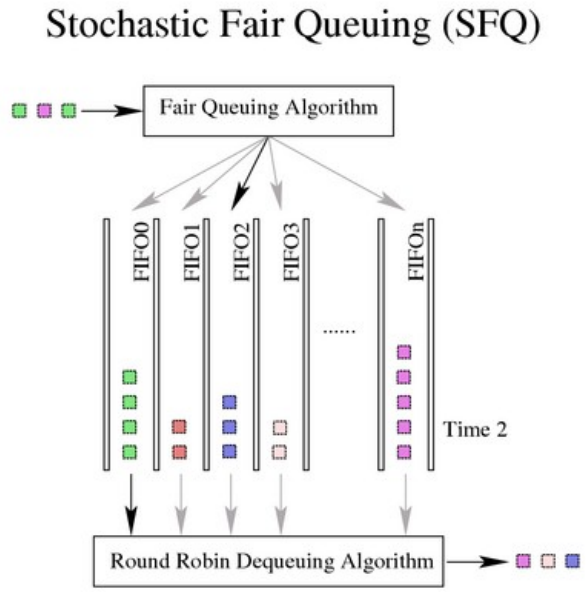


  - ***pfifo_fast*** → Son las *qdiscs* por defecto para todas las interfaces bajo Linux. Consisten en una cola de tres bandas (FIFOS individuales) para separar el tráfico. El tráfico con mayor prioridad (flujos interactivos) es colocado en la banda 0 y siempre son servidas primero. De igual modo, la banda 1 debe haber sido vaciada antes de que la banda 2 empiece a desencolar elementos.



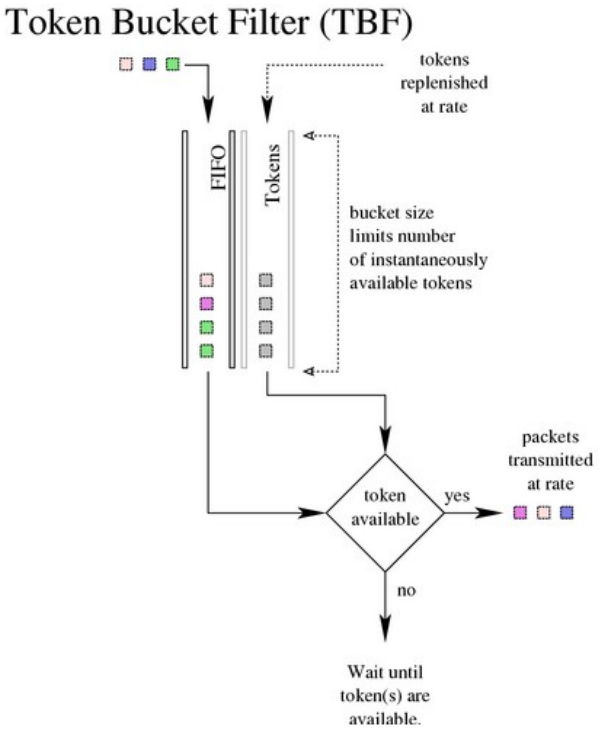
  - ***red*** → La *Random Early Detection (*Detección Aleatoria Temprana) simula la congestión física mediante la reducción aleatoria de paquetes, cuando se configura el ancho de banda. Es muy adecuada para las aplicaciones con un ancho de banda muy grande.

- ***sfq*** → El *Stochastic Fairness Queuing* (Encolado Estocástico Imparcial) intenta distribuir equitativamente la posibilidad de transmitir un paquete a la red entre un número arbitrario de flujos. Esto se logra utilizando una función *hash* para separar el tráfico en distintas FIFOs que son desencoladas siguiendo el algoritmo de prioridad *Round-Robin.* Debido a la posibilidad de que la elección en la función *hash* no sea del todo imparcial, esta función se ve alterada periódicamente.



- ***tbf*** → El *Token Bucket Filter* (Filtro de Cubos de Símbolos) simplemente modela el tráfico transmitido en una interfaz. Esta *qdisc* es la solución perfecta para limitar la velocidad a la que los paquetes salen por una interfaz en particular. Esto lo hace ralentizando el tráfico de transmisión a la velocidad especificada.

  Los paquetes sólo se transmiten si hay símbolos suficientes. De lo contrario su transmisión será aplazada. Retrasar los paquetes de esta manera introduce latencias artificiales en el tiempo de viaje de ida y vuelta del paquete.



- ***qdiscs* con clases:**
  Las *qdiscs* con clases pueden ser:
  - ***CBQ*** → El *Class Based Queuing* (Encolado Basado en Clase) es la implementación clásica de un sistema de control de tráfico. Implementa una jerarquía de clases de enlaces de intercambio. Tiene la capacidad de modelar elementos y de priorizar. El modelado se realiza mediante el cálculo del tiempo de inactividad de un enlace basándose en el tamaño medio del paquete y el ancho de banda del enlace.

  - ***HTB*** → El *Hierarchy Token Bucket* (Cubo de la Jerarquía de Símbolos) usa los conceptos de símbolos y cubos junto con el sistema basado en clases y filtros para permitir el control complejo y granular del tráfico.

    Esta disciplina de colas permite al usuario definir las características de los símbolos y los cubos utilizados y permite al usuario encajar estos cubos de una manera arbitraria. Cuando se combina con un sistema de clasificación, el tráfico puede ser controlado de una manera muy granular.

  - ***PRIO*** → Priority Scheduler (planificación de prioridad). Las *qdiscs* con clases PRIO trabajan de una manera simple. Cuando se está preparado para desencolar un paquete, se comprueba si la primera clase tiene algún paquete. Si tiene, este es desencolado. Si no tiene, se comprobará la siguiente clase; y así sucesivamente. Cada clase contiene a los paquetes que tienen una misma prioridad.

3. Trabajando con “tc”.

Con *tc* podemos realizar las siguientes acciones:

- ***add***: añade una clase, *qdisc* o filtro al nodo correspondiente.
  Se debe tener en cuenta que a la hora de crear cualquiera de estas entidades el nodo que será su padre deberá ser valido, ya sea teniendo un ID válido o bien porque este sea el nodo raíz donde se encuentra el dispositivo asociado. A continuación se ven los comandos completos para la creación de *qdisc*, clases y filtros.

```
tc qdisc add dev (device_name) [parent qdisc-id|root] [handle qdisc-id] qdisc
                        [qdisc specific parameters]

tc class add dev (device_name) parent qdisc-id [classid class-id] qdisc [qdisc
                        specific parameters]

tc filters add  dev (device_name) [ parent qdisc-id | root ] protocol (protocol)
   prio priority filtertype [ filtertype specific parameters ] flowid flow-id
```

- ***del***: elimina el *qdisc* que corresponda con el identificador '*handler*' que se ha indicado.
  Todos los nodos que dependan de él serán eliminadas a su vez, así como sus correspondientes filtros. A continuación se muestra el comando completo para la eliminación de *qdisc*.

```
tc qdisc del dev (device_name) [parent qdisc-id|root] [handle qdisc-id] qdisc
```

- ***change***: permite cambiar las características de las entidades creadas.
  No sirve para mover nodos, es decir, no pueden cambiarse sus padres. A continuación se muestran los comandos completos:

```
tc qdisc change dev (device_name) [parent qdisc-id|root] [handle qdisc-id] qdisc
                        [qdisc specific parameters]

tc class change dev (device_name) parent qdisc-id [ classid class-id ] qdisc
                        [ qdisc specific parameters ]

   tc filters change dev (device_name) [ parent qdisc-id | root ] protocol
(protocol) prio priority filtertype [ filtertype specific parameters ] flowid
                                    flow-id
```

- ***replace***: reemplaza un nodo por otro, realiza una operación (casi) atómica *add/remove* sobre alguna de las entidades existentes.
  Si el nodo que será reemplazado por otro no existe, se creará antes de reemplazarlo.

```
tc qdisc replace dev (device_name) [parent qdisc-id|root] [handle qdisc-id]
                    qdisc [qdisc specific parameters]

tc class replace dev (device_name) parent qdisc-id [ classid class-id ] qdisc
                        [ qdisc specific parameters ]

   tc filters replace dev (device_name) [ parent qdisc-id | root ] protocol
(protocol) prio priority filtertype [ filtertype specific parameters ] flowid
                                    flow-id
```

- ***link***: realiza una sustitución, al igual que *replace*, pero en este caso el nodo debe existir previamente y solo puede utilizarse para *qdisc*. A continuación se muestra la sintaxis completa para este comando:

```
tc qdisc link dev DEV [parent qdisc-id|root] [handle qdisc-id] qdisc [qdisc
                        specific parameters]
```

- ***show***: comando que permite mostrar información sobre los arboles creados, tiene disponible una serie de opciones.
  - -s, -stats, -statistic: muestra estadísticas.
  - -d, -details: muestra en la salida información más detallada sobre las tasas y tamaños de las casillas.
  - -r, -raw: muestra valores de salida como hexadecimales y en bruto (sin unidades) para los ID de 'handler'
  - -p, -pretty: decodifica los filtros de offset y los valores de la máscara en sus equivalentes comandos de filtro basados en TCP / IP.
  - -iec: muestra las tasas en unidades de IEC (es decir, 1 K = 1024).

  A continuación se muestran los comandos completos para *show*:

```
tc [ -s, -d, -r, -p, -iec ] qdisc show [ dev DEV ]
  tc [ -s, -d, -r, -p, -iec ] class show dev DEV
              tc filter show dev DEV
```

Las *qdisc* sin clases sólo pueden depender de la raíz, si no existen *qdisc* con clases de las que puedan depender.

Ejemplo de creación de *qdisc*:

```
tc qdisc add dev ppp0 root tbf rate 220kbit latency 50ms burst 1540
```

Ejemplo de creación de una clase:

```
tc class add dev eth0 parent 1: classid 1:1 htb rate 10 mbit burst 15k
```

Ejemplo de creación de un filtro:

```
tc filter add dev eth0 parent 1:0 protocol ip prio 10 u32 match ip tos 0x10 0xff
                                flowid 1:4
```

Ejemplo para la eliminación que *qdisc*:

```
tc qdisc del dev ra0 root handle 1: prio
```

Ejemplo para visualizar los elementos creados:

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

Como se ha mostrado, la mayoría de los comandos hablan de parámetros *qdisc* específicos, a continuación hablamos de los más destacados o utilizados:

- ***burst***: tamaño del cubo en bytes.
- ***rate***: tasa de velocidad.
- ***latency***: periodo máximo de tiempo que puede pasar entre paquetes.
- ***ceil*** o ***cell***: tiempo que tarda un paquete en ser transmitido sobre un dispositivo está escalonado, y se basa en el tamaño del paquete.
- ***limit***: cantidad de paquetes que serán encolados.
- ***mpu***: tamaño mínimo del paquete.

Ejemplo interesante para mostrar el funcionamiento en clase:

Creamos una *qdisc htb*, dentro de la cual vamos a crear dos subclases *htb* con el mismo ancho de banda.

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

# tc class add dev eth0 parent 1: classid 1:1 htb rate 3125kbps
# tc class add dev eth0 parent 1: classid 1:2 htb rate 3125kbps
```

Una vez que la *qdisc* y las subclases están creadas, se crean filtros para ambas subclases.

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 10 u32 match ip tos 0x10
0xff flowid 1:2


# tc filter add dev eth0 parent 1:0 protocol ip prio 9 u32 match ip tos 0x08
0xff flowid 1:1
```

Referencias:

- http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html
- http://www.linux.org.pe/lnd/PLUG-LND-ControlTraficoLinux26-JSarmiento.pdf
- http://www.gulic.org/almacen/lartc.pdf
- http://www.monografias.com/trabajos17/ancho-de-banda/ancho-de-banda.shtml
- http://gnesis.crysol.org/es/node/692
- http://crysol.org/node/685
- manual Linux sobre *tc*