TP Calidad de Servicio (QoS) / Traffic Shaping

Fecha de Entrega: 04/11/2020.

Objetivo: Conocer los mecanismos de control de tráfico bajo entornos Linux, cómo definir Traffic Shaping Policies y cómo simular las condiciones adversas de una red WAN mediante el módulo *netem*. Categorías ISO: F**C**A**P**S.

Bibliografía

- EVANS, J., FILSFILS, C., 2007, *Deploying IP and MPLS QoS for Multiservice Networks: Theory & Practice.* Morgan Kaufmann.
  - Capítulo 1. "QOS Requirements and Service Level Agreements"
  - Capítulo 2. "Introduction to QOS Mechanics and Architectures"
- MEDHI, D., RAMASAMY, K., 2007, *Network Routing Algorithms Protocols and Architectures.* Morgan Kaufmann.
  - Capítulo 14. "Router Architectures"
  - Capítulo 22. "Packet Queuing and Scheduling"
  - Capítulo 23. "Traffic Conditioning"

Experiencia en el laboratorio

Se realizarán pruebas con utilidades estándar de Linux para apreciar el efecto que causan algunas características no deseadas (pérdidas, delay, reordenamiento) que suelen observarse en redes WAN. El router intermedio entre estos equipos implementará el emulador *netem* en su interfaz de salida hacia el host receptor.

En primera instancia, se debe descargar el laboratorio 'QoS' para Netkit desde este Link y ubicarlo `~/netkit/labs/netkit-lab_qos.tar.gz` .

Luego se debe descomprimir en `~/netkit/netkit-lab_qos` para su uso.

La red del laboratorio es la siguiente:

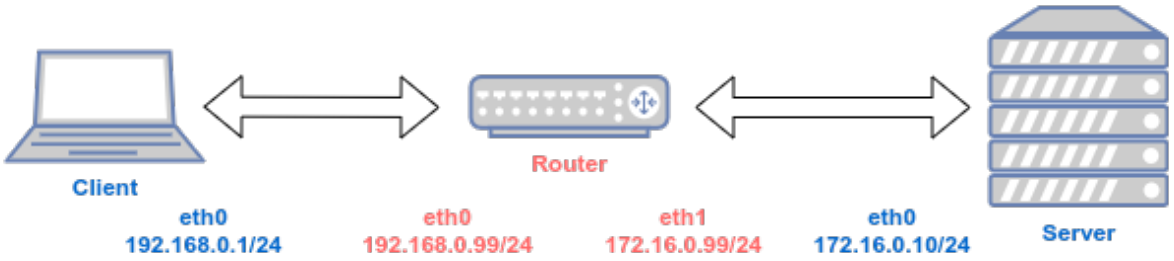

Figura 1: Topología laboratorio Netkit

1. Posicionarse en el directorio `~/netkit/netkit-lab_qos/`
2. Lanzar el laboratorio con el comando `lstart` .
3. Evaluar tasa de transferencia por defecto.

   En el server, ejecutar: `iperf -s -p 80`

   En el cliente, ejecutar: `iperf -c server -p 80`

Con eso se tiene una idea de la tasa de transferencia que hay entre cliente y servidor por medio del router.

4. Evaluar latencia (RTT) por defecto.

   En el server, ejecutar: `ping 192.168.0.1`

   En el cliente, ejecutar: `ping server`

   Con eso se tiene una idea de la latencia que hay entre cliente y servidor por medio del router.

5. Utilizando el emulador netem, modifique en el router las características de la red para simular delay y pérdida de paquetes.

   ```
   # para agregar latencia en el enlace saliente (outbound)
   tc qdisc add dev eth0 root netem delay 200ms

   # para agregar jitter variable
   tc qdisc change dev eth0 root netem delay 200ms 35ms

   # para simular perdida de paquetes
   tc qdisc change dev eth0 root netem loss 5%
   ```

   Recordar que al aplicarlo sobre la interfaz `eth0` del router, este tendrá efecto sobre el la interfaz que apunta la "LAN", o sea, el cliente.

6. Sobre la misma topología, utilizmos TBF (Token Bucket Filter), una disciplina de encolado filtro classless con `tc` para aplicar una política de control de tasa de transferencia de 1Mb/s:

   ```
   # Limitamos la interfaz eth1 del router a 1Mbps, balde de 100 KB
   tc qdisc add dev eth1 root tbf rate 1mbit burst 100kb latency 100ms
   ```

   ¿Por qué se aplica a `eth1` ? (Tip: recordar cómo funciona iperf).

Se recomienda que para la realización de este ejercicio se trabaje con una máquina virtual (Virtualbox, VMWare, etc.) con alguna distribución de Linux (Ubuntu, Debian, etc.), de manera tal de poder instalar los paquetes mencionados (git, graphviz, python3, etc.). De paso, dicha VM podría servir para los TPs "a distancia" que se necesiten en el futuro.

Consignas a resolver

1. Calcule la tasa de transferencia mínima requerida en bps (o múltiplos) y en bytes/seg para las siguientes aplicaciones. Contemple la tasa de bits en crudo, sin compresión alguna y omita el overhead de capas inferiores. ⌂

   a. Aplicación de voz sobre IP punto a punto, donde se transfiere sonido en un solo canal (mono) con una frecuencia de muestreo de 8000 Hz (muestras por segundo) y cada valor es un entero de 8 bits.

   b. Estación meteorológica remota que recopila valores de temperatura, humedad, dirección y velocidad del viento, posición (gps) y remite dichos valores en una PDU a nivel de aplicación de 246 bytes sobre procolo TCP con una periodicidad de 5 segundos. Contemple aquí el overhead de capas inferiores.

c. Aplicación de transmisión de sonido punto a punto con una frecuencia de muestreo de 44100 Hz, donde cada valor de la muestra es un entero entre 0 y 65.535 y se envían dos canales de sonido en simultáneo (izquierdo y derecho).

d. Cámara IP de videovigilancia con una resolución SIF (352 x 288 pixeles) a 25 cuadros por segundo (fps) de video en escala de grises.

e. Aplicación de broadcast (radio en línea), con una frecuencia de muestreo de 22050 Hz, resolución de 16 bits, sonido estéreo y hasta 10 receptores posibles en simultáneo.

f. Aplicación de videollamada simultánea con resolución de video SIF en color de 24 bits (RGB) y sonido con calidad de CD.

g. Servidor de almacenamiento de sonido en un estudio de grabación, almacenando audio a una frecuencia de muestreo de 96 Khz, resolución de 24 bits y 16 canales en simultáneo.

h. Streaming de audio y video digital punto a punto en tiempo real, sin compresión, con resolución de videoResolución Full HD en color RGB (3 x 8 bits) a 60 frames por segundo y audio en 44.1 Khz, 5.1 canales.

Corolario: los valores que surgen de estos cálculos corresponden a flujos en crudo. En las aplicaciones modernas se utilizan codecs de audio y video que optimizan la comunicación, reduciendo notablemente la tasa de transferencia necesaria para las mismas. Sobre esta temática se ampliará en la clase referida a Voz sobre IP.

2. Considerando las siguientes aplicaciones: ⌂

   a. Videojuego con multijugador online. Ejemplo: Overwatch, Call of Duty.

   b. Aplicación de servicio de archivos en la nube: Ejemplo Google Drive, Dropbox

   c. Streaming de video. Ejemplo: Netflix, Flow.

   d. Servicio de DNS.

   e. Servicio de mensajería instantánea. Ejemplo: WhatsApp, Messenger.

   f. Repositorio FTP de imágenes ISO.

   Piense en cómo las aplicaciones se ven afectadas por éstas variables (tasa de transferencia baja/alta, delay, pérdida, duplicación y reordenamiento), así también como por otras de las métricas vistas en clase.

   A partir de allí realice, mediante una tabla, una comparación donde establezca cuáles parámetros o características deberían exigirse para tener una buena calidad de servicio(Ej alta tasa de transferencia, bajo delay, baja perdida, indistinta duplicación, indistinto re ordenamiento). Justifique sus decisiones en cada caso.

3. Se requiere configurar una política de control de tráfico para el uso de un enlace de red que limite la tasa de transferencia independientemente del tipo de tráfico cursado por el enlace. Para este trabajo es necesario configurar una PC como router donde se aplica la política y un cliente y servidor para la generación del tráfico (puede utilizar la herramienta `nc`).

   a. Detalle la configuración realizada para permitir ruteo y la configuración de la política de control aplicada en cada una de las interfaces de red. Verifique la aplicación de la política utilizando las herramientas de medición de tráfico vistas en clase (por ejemplo mediante `iperf` e `iptraf`).

b. Indicar la configuración de `tc` necesaria para aplicar *policing*.
c. Indicar la configuración de `tc` necesaria para aplicar *shaping*.
d. Detalle un ejemplo en que sea necesario aplicar este tipo de políticas.

DESAFIOS

1. El laboratorio de netkit TC netkit-lab_tc.tar.gz simula una red comprendida por el router de borde de una organización, varios servidores, y un cliente externo a la misma.
La capacidad del enlace saliente de la organización es de 5 Mbps. Se ha solicitado establecer una política que destine al tráfico web una tasa mínima garantizada de 2.5 Mbps, al correo electrónico una tasa mínima de 1.5 Mbps, y a la base de datos 512 Kbps, dejando el resto para el tráfico sin clasificar.

   a. Las siguientes líneas establecen la primera parte de la clasificación que se aplica en el router. Complete esta lista de comandos añadiendo las clases y filtros faltantes.

```
# eliminar la disciplina existente
tc qdisc del dev eth0 root

# agregar la disciplina HTB a la raiz del arbol
tc qdisc add dev eth0 root handle 1:0 htb default 8

# crear la clase unica que limita todo el trafico en la interfaz
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 5mbit

# crear la clase 1:5 que sera utilizada para el trafico web
tc class add dev eth0 parent 1:1 classid 1:5 htb rate 2.5mbit

# asignar los paquetes originados en el webserver a la clase 1:5
tc filter add dev eth0 protocol ip parent 1:0 prio 5 u32 \
          match ip src 172.16.0.10/32 flowid 1:5
. . .
```

   b. Descargue e inicie el laboratorio netkit-lab_tc.tar.gz. Cree un archivo de texto con toda la política completa y aplíquela sobre el equipo *router*.

   c. Utilizando los scripts `./test-descarga-serie.sh` y `./test-descarga-paralelo.sh` disponibles en el equipo *cliente*, compruebe que los filtros y clasificaciones aplicados funcionan correctamente. Tenga en cuenta que el throughput está expresado en Mbps.

   d. ¿Los valores de tasa de transferencias obtenidas para cada clase son mayores o menores a los esperados? Brinde una explicación posible para este hecho.

DESAFÍO: Utilizando el componente `tc netem` realice cuatro simulaciones de una red de área amplia (WAN) que reproduzcan los siguientes escenarios: ⌂

a. *Delay variable:* Aplique una regla de delay, definiendo el valor que este adoptará, su variación y un valor de correlación. Detalle la configuración establecida. Utilice una herramienta de diagnóstico de red para evaluar empíricamente la latencia existente y grafique, a partir de las pruebas realizadas, el RTT percibido en función del tiempo.

b. *Pérdida:* Simule un escenario de pérdida de paquetes, luego utilice la herramienta `ping` para verificar que efectivamente las reglas fueron correctamente definidas en el host. De-

talle la configuración establecida. Realice una captura para visualizar el comportamiento y documente lo hallado.

c. *Duplicación:* Simule una situación de duplicación de paquetes, luego utilice la herramienta `ping` para verificar que efectivamente las reglas fueron correctamente definidas en el host. Detalle la configuración establecida. Realice una captura para visualizar el comportamiento y documente lo hallado.

d. *Reordenamiento:* Simule una situación de reordenamiento de paquetes, donde un 25% (con una correlación del 50%) sean enviados inmediatamente mientras que el resto tenga un retraso de 10 ms. Pruebe estableciendo una sesión ssh y verifique el comportamiento a través de una captura.

Referencia

- Páginas del manual
  `man tc` , `tc-netem` , `tc-drr` , `tc-htb` , `tc-filter` , `tc-u32` , ...
  http://man7.org/linux/man-pages/man8/tc.8.html
- Cheat Sheet sobre QoS de PacketLife.net
  http://packetlife.net/media/library/19/QoS.pdf
- Manual tc Packet Filtering and netem. Ariane Keller. ETH Zurich. 2006
  http://tcn.hypert.net/tcmanual.pdf
- Hertzog, R., Mas, R., Capítulo 10.3. Calidad del servicio en “Debian 8: El libro del administrador de Debian”, 2015
  https://debian-handbook.info/browse/es-ES/stable/sect.quality-of-service.html
- Vita Smid, Visualizing Linux Traffic Control Setup (*tcviz*)
  https://github.com/ze-phyr-us/tcviz

Conceptos fundamentales

De acuerdo a su manual, TC maneja **tres** conceptos fundamentales:

QDISCS *qdisc* es la abreviatura de **disciplina de encolado** y es el componente principal para entender el control del tráfico. Siempre que se solicita al sistema operativo enviar un paquete a una interfaz, el kernel lo encola en la *qdisc* que dicha interfaz tiene definida. Inmediatamente después, el kernel trata de desencolar tantos paquetes como sea posible de la *qdisc*, para dárselos al driver de la placa de red.
Una *qdisc* sencilla es la denominada `PFIFO` , que no hace procesamiento y es una cola puramente First-In, First-Out. Desde luego, la qdisc `pfifo` sí almacena el tráfico en caso de que la interfaz de red momentáneamente no lo pueda enviar.
Algunas de las disciplinas de encolado disponibles (`qdisc`) son:

- Simples, sin posibilidad de definir clases (classless)
  - First In, First Out (FIFO)
    - Packet limited First In, First Out (pfifo)
    - Three-band First In, First Out (pfifo_fast) - por defecto en linux antiguos
    - con múltiples colas
    - basadas en hash
      - Stochastic Fairness Queuing (SFQ) - round robin, cada flujo udp o sesión tcp tiene su chance
      - Fair Queuing with Controlled Delay (fq_codel) - por defecto en linux modernos
  - Token Bucket Filter (TBF) - hace shaping o policing según los parámetros
  - Generalized Random Early Detection (GRED) - aplica RED para evitar congestión
  - ...
- Con posibilidad de definir clases (classfull)
  - Hierarchical Token Bucket (HTB)
  - Deficit Round Robin (DRR)
  - Class Based Queueing (CBQ) - antigua y compleja
  - ...

CLASES Algunas *qdisc* pueden contener **clases**, que a su vez pueden contener más qdiscs - el tráfico puede entonces encolarse en cualquiera de las qdisc internas, que están dentro de las clases. Cuando el kernel intenta desencolar un paquete de una qdisc con clases, el mismo puede provenir de cualquiera de las clases hijas (dependiendo de la política). Una qdisc puede, por ejemplo, dar prioridad a ciertos tipos de tráfico al tratar de desencolar paquetes de una cierta clase antes de los otros.

FILTROS Finalmente, un **filtro** es utilizado por una qdisc con clases para determinar en qué clase se encola (o a qué clase corresponde) un paquete que egresará del host. Siempre que el tráfico llega a una clase con subclases, *DEBE ser clasificado*. Se pueden emplear varios métodos, entre los cuales están los filtros. (...) Es importante destacar que los filtros no son globales para toda la interfaz, sino que *siempre residen en una qdisc*.

Teoría de operación

Las clases forman un árbol, donde cada clase tiene un solo padre, pero puede contener varias clases o disciplinas (*qdisc*) hijas. Algunas disciplinas de encolado (por ejemplo `CBQ` y `HTB`) permiten la adición de más clases, mientras que otras (`PRIO`) se crean con un número fijo de clases hijas. Aquellas qdiscs que permiten agregar dinámicamente clases pueden tener cero o más subclases sobre las cuales se puede encolar tráfico.

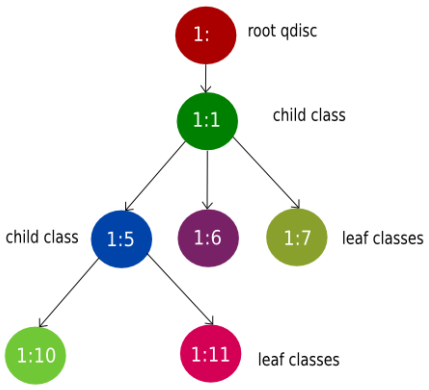

Figura 2: Ejemplo de árbol de clases en tc

Por otra parte, cada **clase** contiene una **qdisc** hoja que por defecto es `PFIFO` o `PFIFO_FAST`, aunque se puede conectar otra qdisc en reemplazo (y así sucesivamente).

Cuando un paquete entra en una qdisc con clases, se puede clasificar en alguna de las clases internas. Hay tres formas distintas de clasificar paquetes, aunque no todas las qdisc las implementan:

- Mediante los **filtros de tc**: si se asocian filtros de tc a una clase, son utilizados primero para determinar lo sucesivo. Los filtros pueden considerar cualquier campo de la cabecera de un paquete [utilizando `u32` o similares], como también las marcas que se ponen por iptables (utilizando `iptables -t mangle ... -j MARK --set-mark NN`).
- Mediante el **tipo de servicio**: algunas qdisc tienen criterios de clasificación propios que se basan en el campo TOS de IP.
- Mediante **skb->priority**: las aplicaciones de usuario que operan contra un socket pueden indicar una id de clase para cada paquete (que tiene vida sólo en el sistema operativo, hasta que el paquete deja el equipo) mediante la opción `SO_PRIORITY`.

Cada nodo en el árbol puede tener sus propios filtros pero los filtros de nivel superior también dirigir directamente a las clases más bajas.

Sintaxis de los comandos

```
# para ver configuración actual y estadísticas
# (tokens disponibles, paquetes descartados, reordenados, etc)
tc -s [ qdisc | class | filter ] show dev INTERFAZ

# para agregar / cambiar / quitar / ver disciplinas de encolado
tc qdisc [ add | change | del ] dev INTERFAZ root nombre_qdisc parametros_qdisc
# para agregar / cambiar / quitar / ver clases de trafico
tc class [ add | change | show ] dev INTERFAZ parent id_qdisc_padre nombre_qdisc
# para agregar / cambiar / quitar / ver reglas de filtrado de trafico
tc filter [ add | change | show ] dev INTERFAZ root protocolo proto prio \
  prioridad tipo_filtro parametros_filtro

# para volver todo a la normalidad
tc qdisc del dev eth0 root
```