



Universidad Rey Juan Carlos

E.T.S. INGENIERÍA DE TELECOMUNICACIÓN

REDES DE ORDENADORES PARA ROBOTS Y MÁQUINAS INTELIGENTES

Práctica 4

Autor:
Javier Izquierdo Hernández

March 18, 2024

Contenidos

1	Control de Tráfico	3
1.1	Sin control de tráfico ni a la entrada ni a la salida	3
1.1.1	Un flujo de datos	3
1.1.2	Dos flujos de datos	4
1.2	Control de admisión para el tráfico de entrada	5
1.3	Disciplinas de colas para el tráfico de salida	8
1.3.1	Token Bucket Filter (TBF)	8
1.3.2	TBF + PRIO	12
1.3.3	Hierarchical token Bucket (HTB)	14
2	Diffserv	18
2.1	Configuración de función policing y marcado de tráfico en DSCP . .	19
2.2	Tratamiento de tráfico en función del marcado DSCP	22
3	Scripts	26

Descarga tu escenario de red para esta práctica del siguiente enlace:

<https://mobiquo.gsync.urjc.es/practicas/ror/p4.html>

Descomprime el fichero que contiene el escenario de NetGUI lab-tc.tgz.

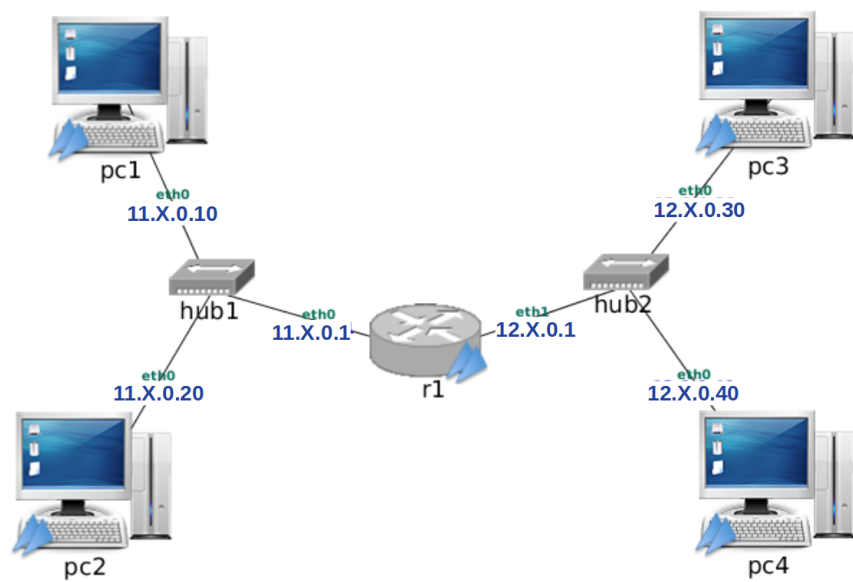


Figure 1: Escenario para control de tráfico

1 Control de Tráfico

1.1 Sin control de tráfico ni a la entrada ni a la salida

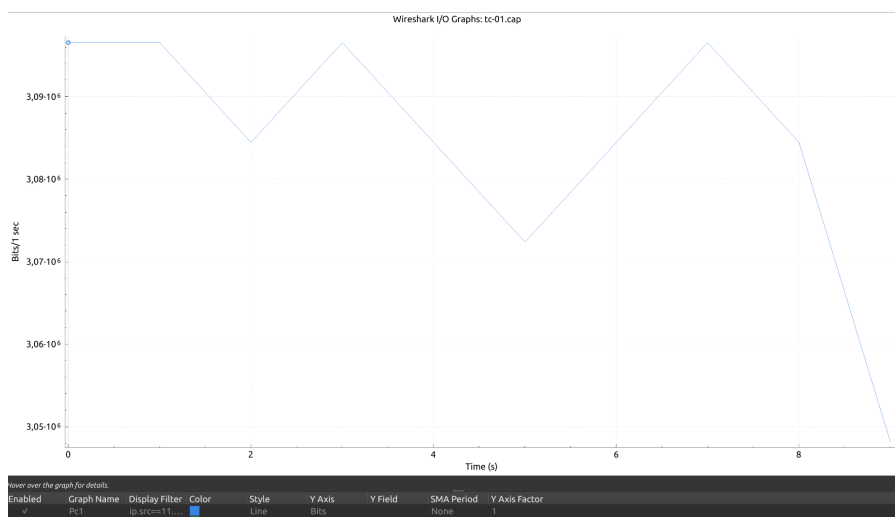
El router r1 no tiene activado el control de tráfico en ninguna de sus interfaces.

1.1.1 Un flujo de datos

Inicia una captura en la interfaz r1(eth1) guardando el contenido en [tc-01.cap](#).

Arranca iperf en modo servidor UDP en pc3 y arranca iperf en modo cliente UDP en pc1 para que envíe tráfico a 3M durante 10 segundos a pc3.

Observa en el servidor el informe que aparece al terminar de recibir el tráfico pc1 → pc3. Carga la captura en wireshark y muestra el flujo de forma gráfica, incluye una imagen en la memoria.



1.1.2 Dos flujos de datos

- Arranca iperf en modo servidor UDP en pc4.
- Arranca otro iperf en modo servidor UDP en pc3.
- Inicia una captura de tráfico en la interfaz eth1 de r1 ([tc-02.cap](#)).
- Escribe (todavía sin ejecutar) el comando que arranca iperf en modo cliente UDP en pc1 para que envíe 3M al servidor pc3 en el sentido pc1 → pc3 durante 10 segundos.
- Escribe (todavía sin ejecutar) el comando que arranca iperf en modo cliente UDP en pc2 para que envíe 3M al servidor pc4 en el sentido pc2 → pc4 durante 10 segundos.
- Ejecuta los dos comandos anteriores uno a continuación de otro (lo más rápidamente que puedas) para que su ejecución se realice de forma simultánea.
- Interrumpe la captura una vez que los clientes hayan terminado de ejecutar iperf.

A continuación analiza los resultados obtenidos:

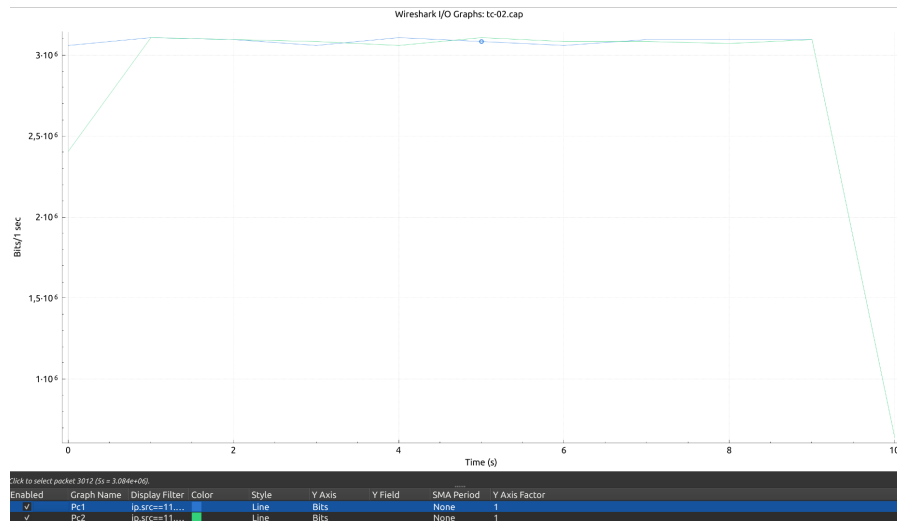
1. Explica las estadísticas que muestran los servidores.

Como esta puesta la política de cola por defecto, ambos servidores reciben la misma cantidad de datos que envían los clientes.

```
pc3:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.30 port 5001 connected with 11.209.0.10 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-10.0 sec  3.58 MBytes  3.00 Mbits/sec  0.129 ms    0/ 2550 (0%)
[ 3] 0.0-10.0 sec  1 datagrams received out-of-order

pc4:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.40 port 5001 connected with 11.209.0.20 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-10.0 sec  3.57 MBytes  3.00 Mbits/sec  0.125 ms    1/ 2549 (0.039%)
[ 3] 0.0-10.0 sec  1 datagrams received out-of-order
```

2. Carga la captura en wireshark y muestra cada uno de los flujos de forma gráfica. Incluye una imagen en la memoria que muestre los flujos de forma gráfica. Explica el ancho de banda medido para cada uno de los flujos.



El ancho de banda será de 3 M/s porque es la cantidad que hemos indicado y como se usa la política de cola por defecto, los datagramas pasan sin problema alguno.

1.2 Control de admisión para el tráfico de entrada

Vamos a configurar r1 para restringir el tráfico de entrada distinguiendo 2 flujos de datos:

- Flujo 1: origen pc1, se quiere restringir con TBF a una velocidad de 1 mbit y una cubeta de 10k bytes.
- Flujo 2: origen pc2 se va a restringir a una velocidad de 2 mbit y una cubeta de 10k bytes.

Utiliza tc para definir esta configuración en la interfaz eth0 de r1, que es la interfaz de entrada para los flujos 1 y 2. Haz que se aplique primero el filtro del flujo número 1 y después el del número 2. Guarda esta configuración en un fichero de script con el nombre [tc-ingress.sh](#):

```
#!/bin/sh

# Esto es un comentario
echo "Borrando la disciplina de cola ingress en la interfaz eth0"

tc qdisc del ...
echo "Creando la disciplina de cola ingress en la interfaz eth0"
tc qdisc add ...
...
```

Una vez creado el script recuerda darle permisos de ejecución.

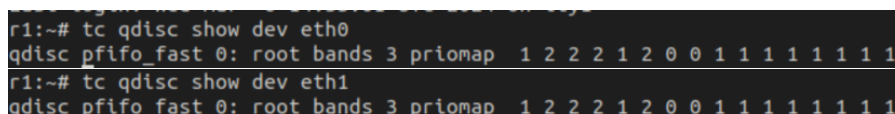
Si prefieres, puedes editar el script en la máquina real (Ubuntu) con un editor gráfico y luego ejecutarlo.

IMPORTANTE: Escribe el script de forma que sólo borre la disciplina de cola si está definida, para que no dé un error. Para ello, utiliza el comando `tc qdisc show dev eth0` y comprueba su salida. Si no devuelve nada, es que no hay ninguna qdisc definida, si devuelve algo es que hay una definida y conviene borrarla primero antes de añadirla.

Incluye dentro de la memoria el contenido del script, así como de los scripts que desarrolles en los siguientes apartados.

Ver 3.1, 3.2, 3.3 y 3.4.

1. Consulta la configuración actual de las disciplinas de cola configuradas por defecto en r1. Indica qué resultado has obtenido para cada una de las colas.



```
r1:~# tc qdisc show dev eth0
qdisc pfifo_fast 0: root bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
r1:~# tc qdisc show dev eth1
qdisc pfifo_fast 0: root bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
```

He obtenido el mismo resultado para ambas, una priomap, que es la cola que viene por defecto.

2. Realiza una prueba de tráfico como la del apartado anterior:
 - Inicia una captura de tráfico en la interfaz eth1 de r1 ([tc-03.cap](#))
 - Arranca dos clientes y 2 servidores tal y como lo hiciste en el apartado 1.1.2.

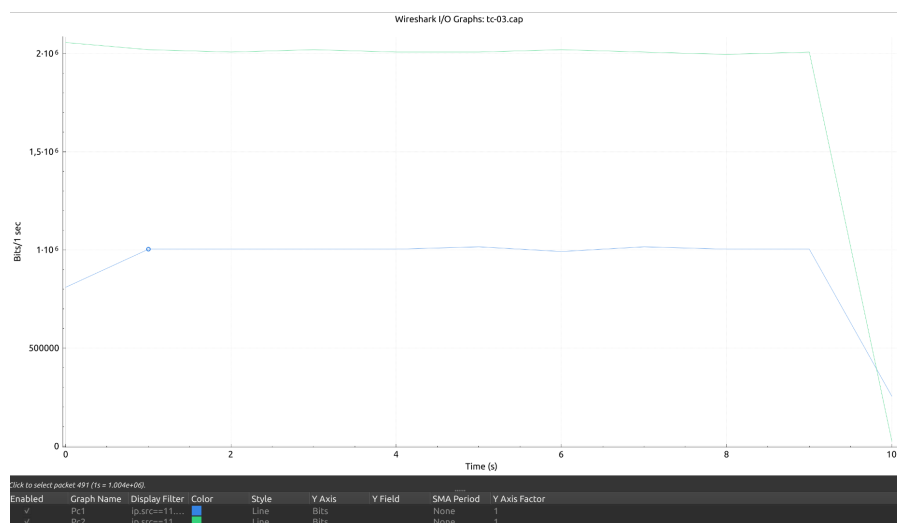
- Interrumpe las capturas cuando los servidores hayan terminado de recibir todo el tráfico.
3. Explica las estadísticas que muestran los servidores.
- Como se limita el flujo de pc1 a pc3 a 1 mbit/s, se deben perder 2/3 de los 3 mbit/s que manda pc1 como se ve en la imagen inferior.

```
pc3:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  108 KByte (default)
-----
[  3] local 12.209.0.30 port 5001 connected with 11.209.0.10 port 32768
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[  3]  0.0-10.2 sec  1.17 MBytes  960 Kbits/sec  15.613 ms 1714/ 2549 (67%)
```

Como se limita el flujo de pc2 a pc4 a 2 mbit/s, se deben perder 1/3 de los 3 mbit/s que manda pc1 como se ve en la imagen inferior.

```
pc4:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  108 KByte (default)
-----
[  3] local 12.209.0.40 port 5001 connected with 11.209.0.20 port 32768
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[  3]  0.0-10.0 sec  2.34 MBytes  1.96 Mbits/sec  0.150 ms  883/ 2550 (35%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
```

4. Carga las capturas en wireshark y muestra cada uno de los flujos de forma gráfica. Incluye una imagen en la memoria que muestre los flujos de ambas capturas de forma gráfica. Explica el ancho de banda medido para cada uno de los flujos.



Como he explicado en el apartado anterior y como se puede ver en la captura, el flujo de datos que manda pc1 se capta a 1 mbit/s, mientras que el de pc2 se capta a 2 mbit/s. Y el resto del tráfico se descarta.

5. Consulta la configuración actual de las disciplina de cola configurada a la entrada en eth0. Indica el número de paquetes recibidos y el número de paquetes descartados.

La configuración de la cola configurada en eth0, es la que hemos puesto en el script, aunque sigue siendo una priomap.

```
r1:~# tc qdisc show dev eth0
qdisc pfifo_fast 0: root bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1
qdisc ingress ffff: parent ffff:fff1 -----
```

Como se puede ver en las imágenes del apartado 3, pc3 recibe 835 y descarta 1714 de un total de 2549, lo que hace un total de un 33% de datagramas recibidos. En cuanto a pc4, este recibe 1667 y descarta 883 de un total de 2549, lo que hace un total de un 65% de datagramas recibidos.

1.3 Disciplinas de colas para el tráfico de salida

1.3.1 Token Bucket Filter (TBF)

Mantén la configuración del tráfico de entrada en r1 que has realizado en el apartado anterior en el script tc-ingress.sh.

- Define en r1 para su interfaz eth1 una disciplina TBF de salida con tasa de envío de 1.5 mbit, tamaño de cubeta 10k y latencia 10 ms, y guarda la configuración en un nuevo script [tc-egress-tbf.sh](#).
- Inicia una captura de tráfico en la interfaz eth1 de r1 ([tc-04.cap](#)).
- Arranca 2 clientes y 2 servidores tal y como lo hiciste en el apartado 1.1.2.
- Interrumpe la captura cuando los servidores hayan terminado de recibir todo el tráfico.

A continuación analiza los resultados obtenidos:

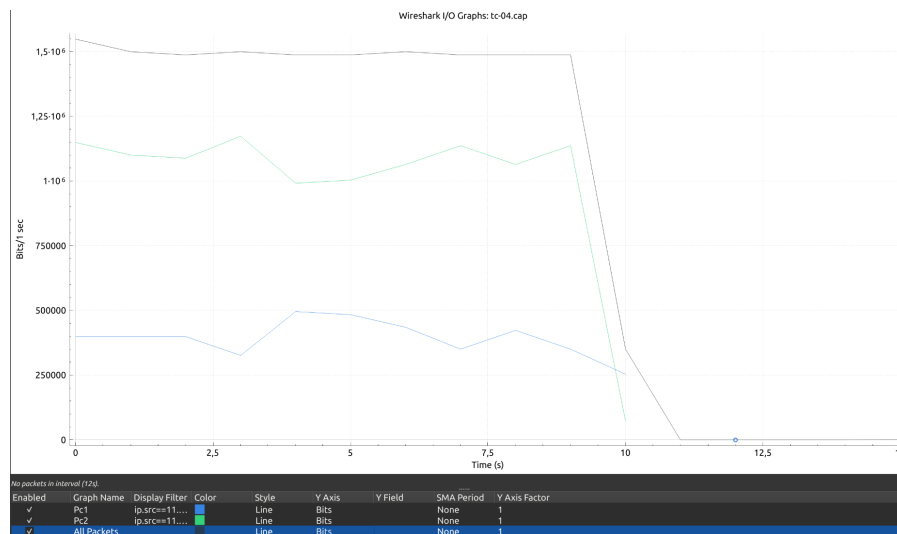
1. Explica las estadísticas que muestran los servidores.

Como se limita el flujo de pc1 a 1 mbit/s y el de pc2 a 2 mbit/s, se puede ver que pc4 deberá recibir el doble que pc3 aproximadamente. Pero como ahora en r1, se limita la salida a 1.5 mbit, en vez de los 3 mbit/s del apartado anterior se reducirá el flujo de salida a 1.5 mbit/s.

```
pc3:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.30 port 5001 connected with 11.209.0.10 port 32768
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[ 3] 0.0- 9.9 sec   511 KBytes  421 Kbits/sec  3.499 ms  2196/ 2552 (86%)

pc4:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.40 port 5001 connected with 11.209.0.20 port 32768
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[ 3] 0.0-10.2 sec  1.27 MBytes  1.04 Mbits/sec  13.042 ms 1642/ 2549 (64%)
[ 3] 0.0-10.2 sec  1 datagrams received out-of-order
```

2. Carga la captura en wireshark y muestra cada uno de los flujos de forma gráfica. Explica el ancho de banda medido para cada uno de los flujos.



Como he explicado en el apartado anterior y como se puede ver en la captura, el flujo de datos que manda pc1 oscila alrededor de 400-500 kbit/s, mientras que el de pc2 lo hace en torno a 1-1.2 mbit/s. También se puede ver que la suma de ambos flujos se mantiene estable en el tope 1.5 mbit/s, y que acaba con un poco de delay gracias a los 10 ms puestos. Al igual que anteriormente el resto de mensajes se descartan.

Modifica la configuración de TBF de salida para que ahora tenga una latencia de 20 segundos (NOTA: ahora son 20 segundos en vez de 10 milisegundos) y realiza la misma prueba que antes 2 . Llama ahora a la captura [tc-05.cap](#).

Interrumpe la captura cuando haya pasado tiempo suficiente para que termine de llegar todo el tráfico a los servidores (será unos 20 segundos después de que comenzó a enviarse). A continuación analiza los resultados obtenidos:

3. Explica las estadísticas que muestran los servidores.

Las estadísticas se pueden explicar de la misma manera que en el apartado 1, con la excepción de que ahora en vez de ser el intervalo entre 0 y 10, ahora será de 0 a 20 segundos. También es importante recordar que estas cantidades de transferencia son las mismas que solo con el filtro en la entrada, pero con el ancho de banda reducido a la mitad, ya que el tiempo se dobla.

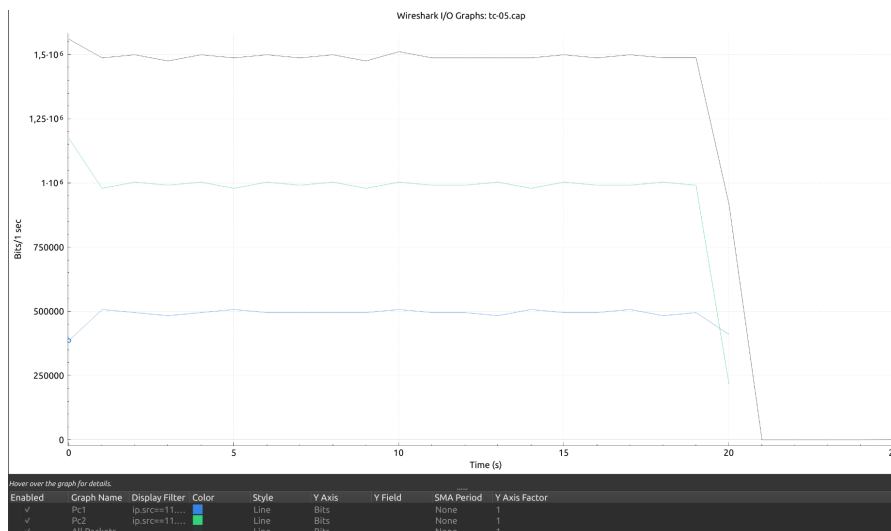
```

pc3:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.30 port 5001 connected with 11.209.0.10 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-20.0 sec  1.17 MBytes  493 Kbits/sec  7.677 ms 1713/ 2551 (67%)
[ 3] 0.0-20.0 sec  1 datagrams received out-of-order
read failed: Connection refused

pc4:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.40 port 5001 connected with 11.209.0.20 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-20.1 sec  2.34 MBytes  975 Kbits/sec  7.165 ms  883/ 2550 (35%)
[ 3] 0.0-20.1 sec  1 datagrams received out-of-order
read failed: Connection refused

```

4. Carga la captura en wireshark y muestra cada uno de los flujos de forma gráfica. Incluye una imagen en la memoria que muestre los flujos de forma gráfica. Explica el ancho de banda medido para cada uno de los flujos.



Al igual que en el apartado 2, pero esta vez en vez de descartar el resto de paquetes justo después de que pasaran 10 ms, ahora se espera hasta los 20 segundos, que es cuando todo el tráfico que ha entrado a r1 es enviado.

5. ¿Cuánto tiempo ha tardado r1 en realizar el reenvío de todo el tráfico? Relaciona este valor con la cantidad de datos que tenía que reenviar y la tasa de envío que estaba utilizando r1. Del tráfico originalmente enviado por pc1 y pc2, ¿cuánto ha sido descartado en la disciplina de cola asociada a eth0 de

r1? ¿Y cuánto ha sido descartado en la disciplina asociada a eth1 de r1?

R1 ha tardado 20 segundos en realizar el reenvío, que es el doble del tiempo durante el que envían pc1 y pc2, ya que el filtro de salida reduce la salida de 3 mbit/s a 1.5 mbit/s.

Luego el tráfico de pc1 y pc2 solo ha sido descartado por la disciplina de la cola asociada con eth0, ya que se ha enviado el mismo tráfico que cuando solo estaba esta.

1.3.2 TBF + PRIO

Mantén la configuración del tráfico de entrada en r1 que has realizado en el apartado anterior en el script `tc-ingress.sh`. Borra la disciplina de cola de salida TBF configurada en la interfaz eth1 de r1.

La configuración TBF en el apartado 1.3.1 permite gestionar la tasa de envío para que no supere el valor configurado, en nuestro caso 1.5Mbit. Esta disciplina de cola es sin clases y trata a todos los paquetes por igual. Ahora vamos a querer fijar la tasa de envío de r1 pero tratando los paquetes con diferentes prioridades.

Toma como punto de partida esta configuración para que ahora se atienda el tráfico de salida según diferentes prioridades, configurando una disciplina de cola con prioridad que sea hija de la disciplina TBF.

- Escribe un script en r1, [tc-egress-tbf-prio.sh](#), para configurar TBF con los siguientes parámetros: ancho de banda 1.5 mbit, cubeta 10k y latencia 20s. Crea una disciplina de cola hija con prioridad de tal forma que se asignen las siguientes prioridades:
 - Prioridad 1 (más prioritario): tráfico de la dirección IP origen pc1
 - Prioridad 2 (prioridad intermedia): tráfico de la dirección IP origen pc2
 - Prioridad 3 (menos prioritario): sin definir, pues no lo necesitamos.
- Inicia una captura de tráfico en la interfaz eth1 de r1 ([tc-06.cap](#)).
- Arranca 2 clientes y 2 servidores tal y como lo hiciste en el apartado 1.1.2.
- Interrumpe la captura cuando haya pasado tiempo suficiente para que termine de llegar todo el tráfico a los servidores (será unos 20 segundos después de que comenzó a enviarse).

A continuación analiza los resultados obtenidos:

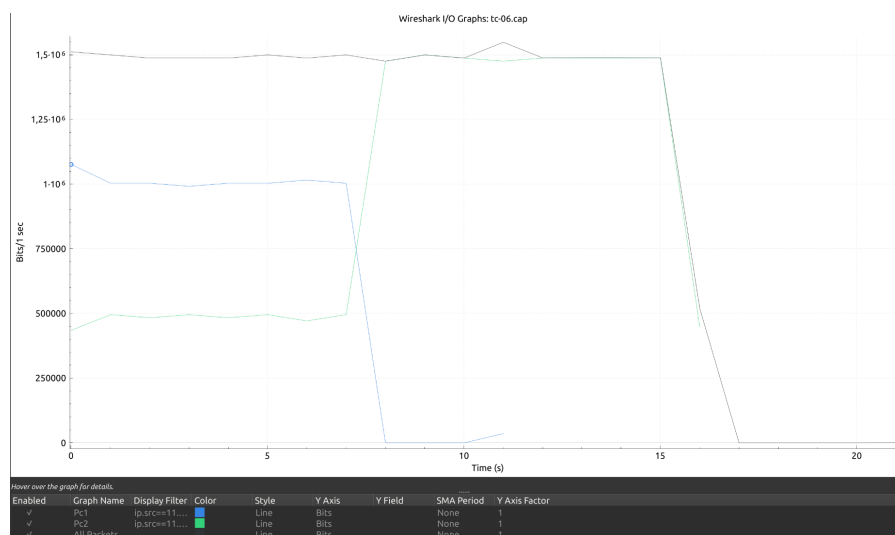
1. Explica las estadísticas que muestran los servidores.

Como pc1 tiene más prioridad acaba antes que pc2, y por lo tanto su ancho de banda será mayor que antes cuando no tenía prioridad.

```
pc3:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.30 port 5001 connected with 11.209.0.10 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-11.9 sec   962 KBytes  663 Kbits/sec 120.857 ms 1881/ 2551 (74%)

pc4:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.40 port 5001 connected with 11.209.0.20 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-16.0 sec   1.87 MBytes  978 Kbits/sec 127.992 ms 1216/ 2550 (48%)
read failed: Connection refused
```

2. Carga la captura en wireshark y muestra cada uno de los flujos de forma gráfica. Incluye una imagen en la memoria que muestre los flujos de forma gráfica. Explica la evolución en el tiempo del ancho de banda medido para cada uno de los flujos.



Como el flujo desde pc1 tiene prioridad y esta capado en la entrada a 1 mbit/s, este será siempre 1 mientras que envíe mensajes, mientras que pc2 será 0.5

mbit/s (el resto) siempre que pc1 envíe el flujo de datos.
Luego cuando pc1 no este, el flujo de datos desde pc2 se enviará ocupando todo el ancho de banda disponible, 1.5 mbit/s.

1.3.3 Hierarchical token Bucket (HTB)

Mantén la configuración del tráfico de entrada en r1 que has realizado en el apartado anterior en el script tc-ingress.sh. Borra la disciplina de cola de salida configurada en la interfaz eth1 de r1.

- Escribe un script en r1, [tc-egress-htb.sh](#), para configurar en su interfaz eth1 una disciplina HTB de salida con ancho de banda 1.2 mbit. Reparte el ancho de banda de esta interfaz de salida de la siguiente forma:
 - 700kbit para el tráfico con origen en pc1, ceil 700kbit.
 - 500kbit para el tráfico con origen en pc2, ceil 500kbit.
- Inicia una captura de tráfico en la interfaz eth1 de r1 ([tc-07.cap](#)).
- Arranca 2 clientes y 2 servidores tal y como lo hiciste en el apartado 1.1.2.
- Interrumpe la captura cuando haya pasado tiempo suficiente para que termine de llegar todo el tráfico a los servidores (será unos 20 segundos después de que comenzó a enviarse). iperf.

A continuación analiza los resultados obtenidos:

1. Explica las estadísticas que muestran los servidores.

Como el flujo de pc1 se capta 700 kbit/s el ancho de banda será ese, mientras que la transferencia será de todos los datos que tenía que enviar. Y la duración del envío aumenta de 10 a 14.5 segundos.

Algo similar ocurre con el flujo de pc2 pero con el ancho de banda siendo 500 kbit/s y el intervalo 34 segundos.

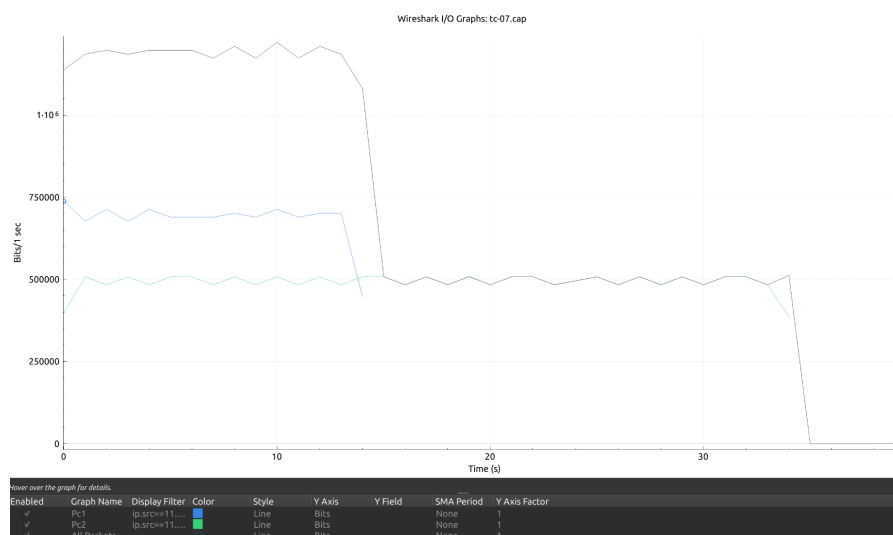
```

pc3:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.30 port 5001 connected with 11.209.0.10 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-14.5 sec  1.17 MBytes  678 Kbits/sec  22.603 ms 1713/ 2550 (67%)
[ 3] 0.0-14.5 sec  1 datagrams received out-of-order
read failed: Connection refused

pc4:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 108 KByte (default)
-----
[ 3] local 12.209.0.40 port 5001 connected with 11.209.0.20 port 32768
[ ID] Interval      Transfer    Bandwidth    Jitter    Lost/Total Datagrams
[ 3] 0.0-34.3 sec  1.98 MBytes  484 Kbits/sec  20.271 ms 1137/ 2550 (45%)
read failed: Connection refused

```

2. Carga la captura en wireshark y muestra cada uno de los flujos de forma gráfica. Explica la evolución en el tiempo del ancho de banda medido para cada uno de los flujos.



Se ve que el flujo máximo que sale de r1 se mantiene estable en 1.2 mbit/s mientras que pc1 envía a 0.7 mbit/s y pc2 envía a 0.5 mbit/s. También se puede apreciar que r1 encola el tráfico hasta reenviarlo todo.

Modifica la configuración de ceil en cada uno de los flujos para que puedan utilizar 1.2Mbit. Realiza la misma prueba que antes capturando de nuevo el tráfico ([tc-08.cap](#)) y analiza los resultados obtenidos:

3. Explica las estadísticas que muestran los servidores.

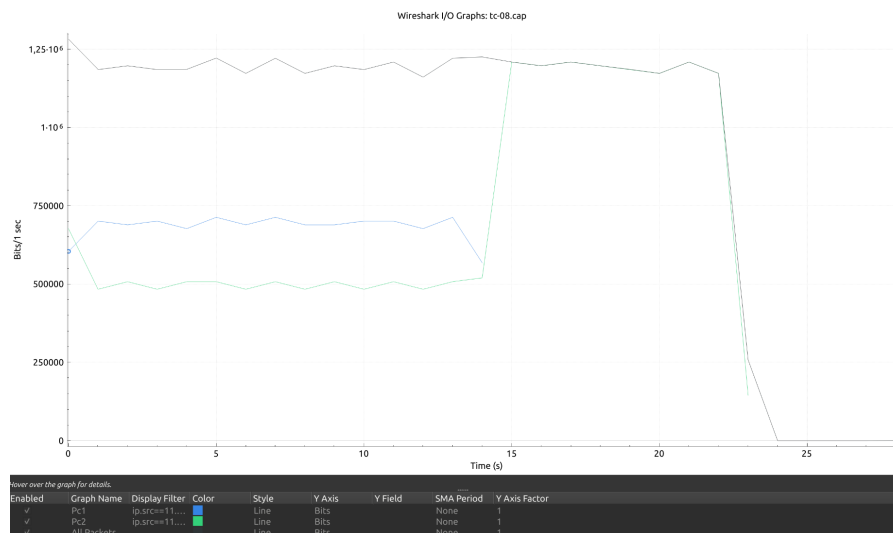
Con el flujo de pc1 ocurre lo mismo que en el apartado anterior, pero con el de pc2 varía.

Este pasa de tardar 34 segundos a 23, y su ancho de banda aumenta.

```
pc3:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  108 KByte (default)
-----
[  3] local 12.209.0.30 port 5001 connected with 11.209.0.10 port 32768
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[  3]  0.0-14.5 sec  1.17 MBytes  678 Kbits/sec  21.337 ms 1712/ 2549 (67%)
read failed: Connection refused

pc4:~# iperf -u -s
-----
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  108 KByte (default)
-----
[  3] local 12.209.0.40 port 5001 connected with 11.209.0.20 port 32768
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[  3]  0.0-23.0 sec  2.00 MBytes  728 Kbits/sec  22.091 ms 1124/ 2550 (44%)
read failed: Connection refused
```

4. Carga la captura en wireshark y muestra cada uno de los flujos de forma gráfica. Incluye una imagen en la memoria que muestre los flujos de forma gráfica. Explica la evolución en el tiempo del ancho de banda medido para cada uno de los flujos.



Durante la parte que esta el flujo de pc1 es igual que en la captura anterior. La diferencia entre ambas es en la parte donde solo esta pc2, ya que en esta, en vez de estar limitada a 0.5 mbit/s, aquí usa todo el ancho de banda disponible.

2 Diffserv

Descarga tu escenario de red para esta práctica del siguiente enlace:

<https://mobiquo.gsync.urjc.es/practicas/ror/p4.html>

Descomprime el fichero que contiene el escenario de NetGUI lab-DiffServ.tgz para realizar la práctica de diffServ en Linux.

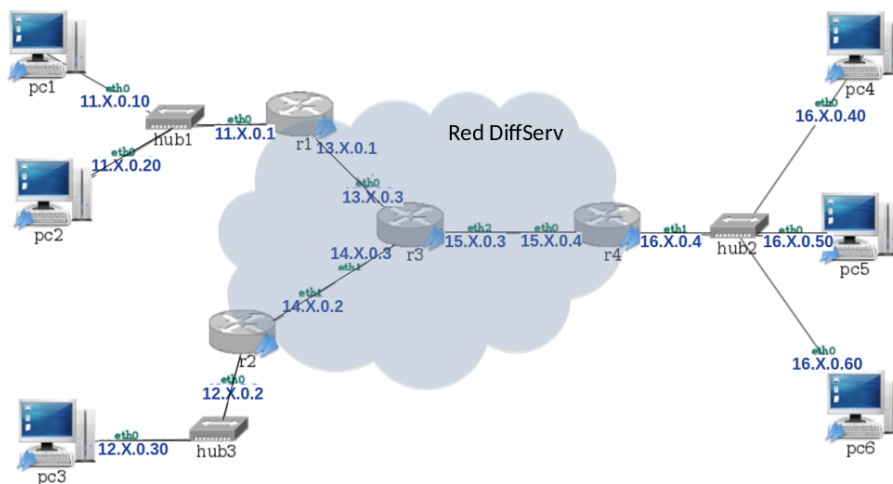


Figure 2.1: Escenario para DiffServ

En el escenario de la figura 2 se va a configurar la red para que el tráfico desde pc1, pc2 y pc3 envíen paquetes a pc4, pc5 y pc6 atravesando una red diffServ. Configura las direcciones IP en tu escenario utilizando las tus 4 subredes de la práctica 1, y elige las subredes que quieras como subred 13.X.0.0/24 y subred 14.X.0.0/24.

Para esta práctica se distinguirán 4 calidades diferentes, con los códigos EF, AF31, AF21 y AF11.

2.1 Configuración de función policing y marcado de tráfico en DSCP

Utiliza la herramienta `tc` para garantizar que el tráfico que entra en `r1` cumple las siguientes características:

- La red `diffServ` deberá garantizar a la entrada los siguientes anchos de banda para `pc1`, descartando el tráfico sobrante:
 - Flujo 1: máximo 1.2mbit con ráfaga 10k para el tráfico dirigido a `pc4`, marcado con calidad EF. Si se supera este ancho de banda, el tráfico quedará clasificado dentro del flujo 2.
 - Flujo 2: máximo de 600kbit y ráfaga 10k, marcado con calidad AF31. Si se supera este ancho de banda, el tráfico será descartado definitivamente en `r1`.
- La red `diffServ` deberá garantizar a la entrada los siguientes anchos de banda para `pc2`, descartando el tráfico sobrante:
 - Flujo 3: máximo 300kbit con ráfaga 10k para el tráfico dirigido a `pc5`, marcado con AF21. Si se supera este ancho de banda, el tráfico quedará clasificado dentro del flujo 4.
 - Flujo 4: máximo de 400kbit y ráfaga 10k, marcado con AF11. Si se supera este ancho de banda, el tráfico será descartado definitivamente en `r1`.

Utiliza la herramienta `tc` para garantizar que el tráfico que entra en `r2` cumple las siguientes características:

- La red `diffServ` deberá garantizar a la entrada los siguientes anchos de banda para `pc3`, descartando el tráfico sobrante:
 - Flujo 5: máximo 400kbit con ráfaga 10k dirigido a `pc6`, marcado con AF31. Si se supera este ancho de banda, el tráfico quedará clasificado dentro del flujo 6.
 - Flujo 6: máximo 300kbit con ráfaga 10k dirigido a `pc6`, marcado con AF21. Si se supera este ancho de banda, el tráfico quedará clasificado dentro del flujo 7.
 - Flujo 7: máximo 100kbit con ráfaga 10k, marcado con AF11. Si se supera este ancho de banda, el tráfico será descartado definitivamente en `r2`.

1. Realiza scripts para r1 y otro para r2 donde se configuren estos perfiles de tráfico. Incluye dichos scripts en la memoria.

Ver 3.5 para r1 y 3.6 para r2.

2. Inicia capturas: [diffServ-01.cap](#) en la subdred 13.X.0.0/24, [diffServ-02.cap](#) en la subdred 14.X.0.0/24 y [diffServ-03.cap](#) en la subdred 15.X.0.0/24 para que capture el tráfico que se genera en tu escenario por el envío "simultáneo" de:

- Desde el pc1 2M a pc4
- Desde el pc2 1.5M a pc5
- Desde el pc3 1M a pc6

3. Interrumpe las capturas, al menos 1 minuto después de que la transmisión haya terminado. Comprueba que el resultado es el esperado:

- El tráfico que entra en la red diffServ es el que se ha especificado en el control de admisión.
- El tráfico está marcado según las especificaciones anteriores.

Para ello, consulta las gráficas IO graphs de Wireshark aplicando los filtros sobre las marcas DSCP de tal forma que se muestre cada calidad marcada de cada una de las fuentes:

- Tráfico de EF
- Tráfico de AF31
 - Total
 - Con origen en pc1.
 - Con origen en pc3.
- Tráfico de AF21
 - Total
 - Con origen en pc2.
 - Con origen en pc3.
- Tráfico de AF11
 - Total
 - Con origen en pc2.
 - Con origen en pc3.

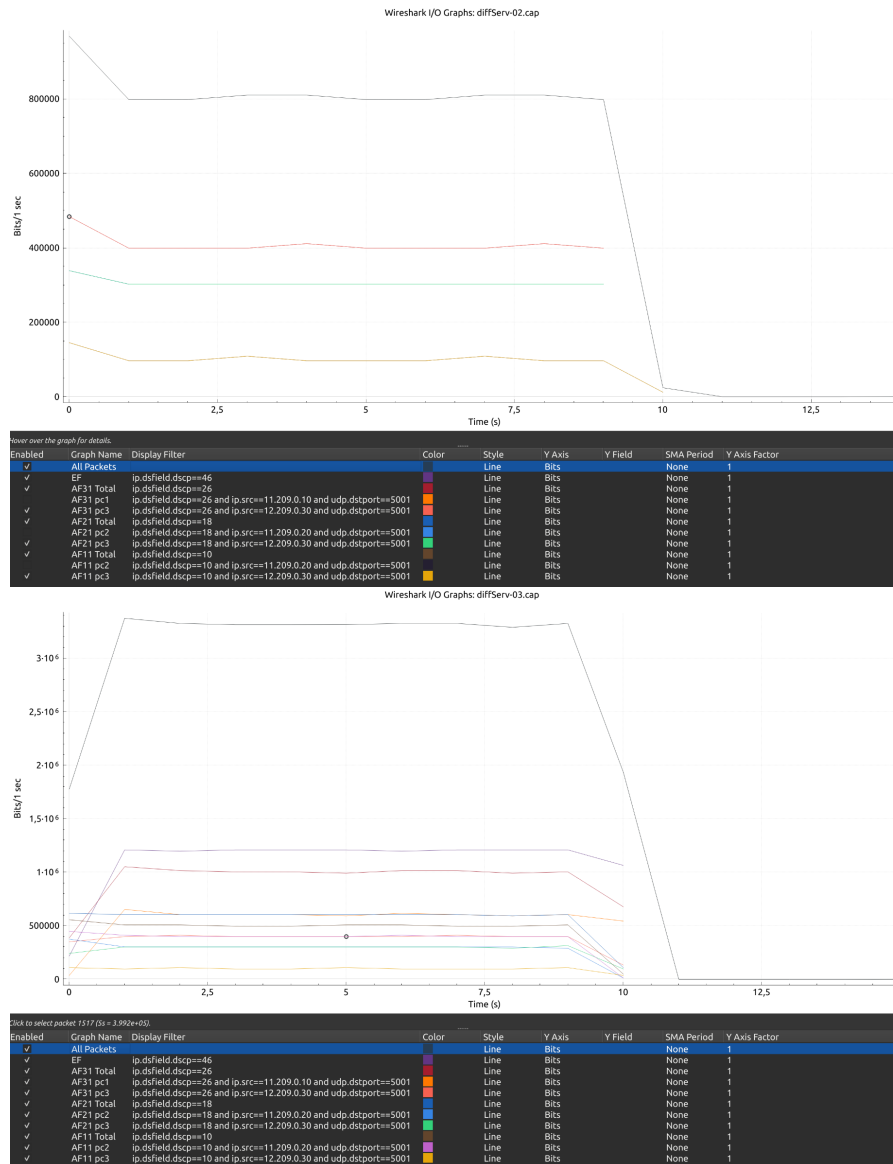
Explica los resultados obtenidos e incluye todas las gráficas que consideres necesarias en la memoria.

Las 2 gráficas importantes son las dos primeras, ya que la tercera es la suma de estas dos.

Luego, en la primera captura se puede ver como pc1 ocupa al máximo los flujos que tiene disponibles y se descarta los 0.2 mbit/s sobrantes. Y con el flujo desde pc2 ocurre lo mismo, ya que este ocupa los dos flujos que tiene disponible (el de AF21 y 11) en su totalidad, descartando 0.8 mbit/s de datos.



En la segunda captura solo se ven los flujos que utiliza pc3, que son AF31, 21 y 11, y los utiliza también en su totalidad, por lo que el resto de datos, 0.2 mbit/s, se descartan.



2.2 Tratamiento de tráfico en función del marcado DSCP

Mantén la configuración realizada en r1, r2.

Se establecen los siguientes parámetros de calidad dentro del router del núcleo diffServ (r3) para cada una de las calidades definidas. Configura HTB con ancho de banda 2.4Mbit para compartir entre todos los flujos con el siguiente patrón:

- EF: HTB 1Mbit como mínimo y 1Mbit como máximo.
 - AF31: HTB 500kbit como mínimo y 500kbit como máximo.
 - AF21: HTB 400kbit como mínimo y 400kbit como máximo.
 - AF11: HTB 200kbit como mínimo y 200kbit como máximo.
1. Realiza un script para r3 donde se configure esta disciplina de cola según el marcado de los paquetes e incluye dicho script en la memoria.

Ver 3.7.

2. Inicia una captura ([diffServ-04.cap](#)) en la subdred 15.X.0.0/24 para que capture el tráfico que se genera en tu escenario por el envío "simultáneo" de:
 - Desde pc1: 2M a pc4
 - Desde pc2: 1.5M a pc5
 - Desde pc3: 1M a pc6

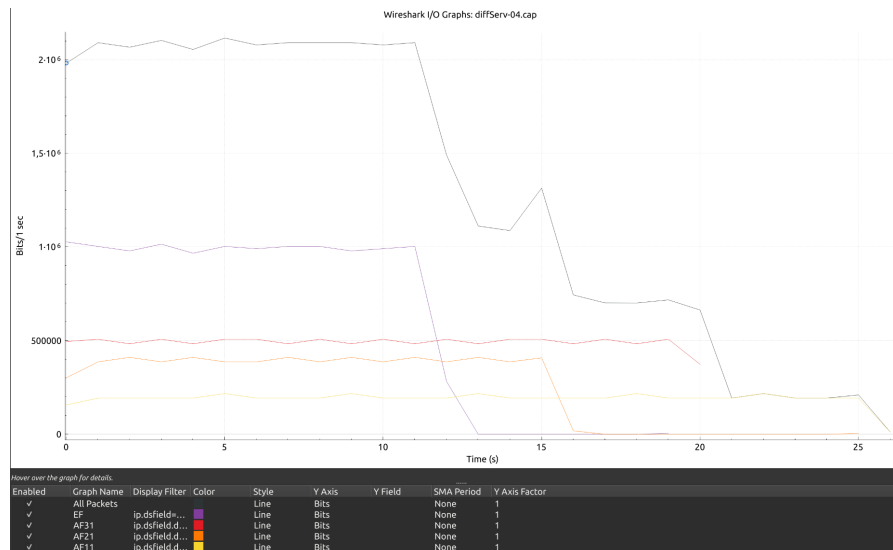
Espera al menos 2 minutos después de que haya terminado de enviarse el tráfico de pc1, pc2 y pc3 antes de interrumpir la captura de tráfico.

3. Comprueba que el resultado es el esperado, es decir, el tráfico sigue el perfil indicado en las especificaciones anteriores. Para ello, consulta las gráficas IO graphs de Wireshark aplicando los filtros sobre las marcas DSCP de tal forma que se muestre cada calidad marcada de cada una de las fuentes incluyendo dichas imágenes en la memoria:
 - Tráfico de EF
 - Tráfico de AF31
 - Tráfico de AF21
 - Tráfico de AF11

Explica los resultados obtenidos y explica si alguno de los flujos ha encolado tráfico para enviarlo posteriormente a los 10 segundos que dura la transmisión de iperf.

Es bastante fácil de ver que todos los flujos encolan tráfico para su posterior reenvío, ya que envían mensajes pasados los 10 segundos que dura la transmisión. Esto también se puede ver al comparar numéricamente el flujo que

entraba en la última captura del apartado anterior con un nuevo filtro de salida.



- Modifica la configuración de HTB en r3 para que si algún flujo no está utilizando el ancho de banda que tiene garantizado lo puedan usar el resto de flujos y vuelve a hacer una captura de tráfico ([diffServ-05.cap](#)) en la subred 15.X.0.0/24. Explica qué modificaciones has tenido que hacer en el script.

Las modificaciones han sido cambiar el ceil de los flujos por el ceil máximo. Ver 3.7.

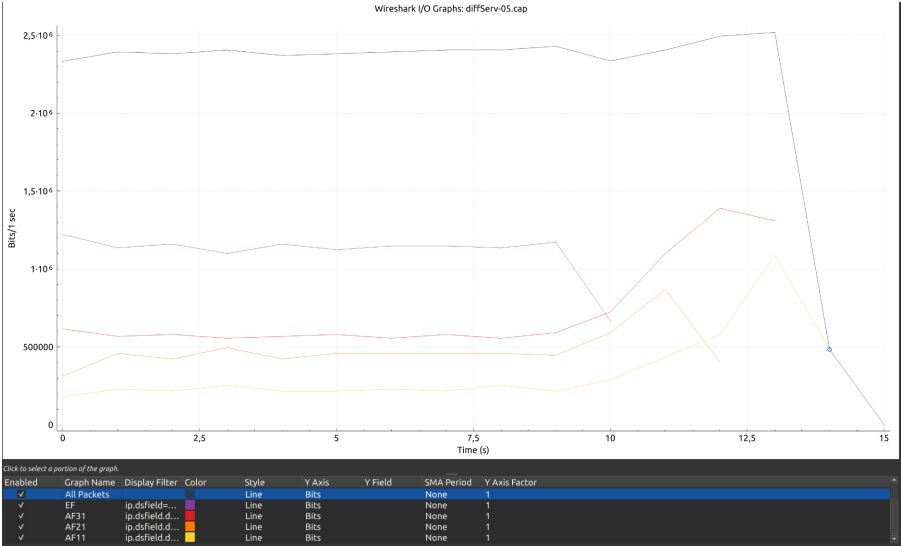
- Explica los resultados obtenidos e incluye las gráficas IO graphs que consideres necesarias.

En esta captura se puede ver como el flujo de datos totales se mantiene constante en el flujo máximo de 2.4 mbit/s y que ya no es necesario encolar todo el tráfico.

El procedente de EF no se encola ya que tiene mayor prioridad, y ahora reenvía datos a 1.2 mbit/s en vez de a 1 como anteriormente.

Con el resto de flujos ocurre algo similar pero siguen encolando el tráfico, y también se ve que cuando acaba un flujo con mayor prioridad estos sufren un aumento en el ancho de banda.

Y por último el tiempo total se ve reducido en 10 segundos respecto a los 25 del apartado anterior, al usar de manera más eficiente el flujo total.



3 Scripts

Listing 3.1: tc-ingress.sh

```
#!/bin/sh

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

tc filter add dev eth0 parent ffff: \
    protocol ip prio 4 u32 \
    match ip src 11.209.0.10/32 \
    police rate 1mbit burst 10k drop flowid :1

tc filter add dev eth0 parent ffff: \
    protocol ip prio 5 u32 \
    match ip src 11.209.0.20/32 \
    police rate 2mbit burst 10k drop flowid :2
```

Listing 3.2: tc-egress-tbh.sh

```
#!/bin/sh

tc qdisc del dev eth0 ingress
tc qdisc del dev eth1 root

tc qdisc add dev eth0 ingress handle ffff:

tc filter add dev eth0 parent ffff: \
    protocol ip prio 4 u32 \
    match ip src 11.209.0.10/32 \
    police rate 1mbit burst 10k drop flowid :1

tc filter add dev eth0 parent ffff: \
    protocol ip prio 5 u32 \
    match ip src 11.209.0.20/32 \
    police rate 2mbit burst 10k drop flowid :2

tc qdisc add dev eth1 root handle 1: tbf rate 1.5Mbit burst 10k latency 10ms
# tc qdisc add dev eth1 root handle 1: tbf rate 1.5Mbit burst 10k latency 20s
```

Listing 3.3: tc-egress-tbh-prio.sh

```
#!/bin/sh

tc qdisc del dev eth0 ingress
tc qdisc del dev eth1 root

tc qdisc add dev eth0 ingress handle ffff:

tc filter add dev eth0 parent ffff: \
    protocol ip prio 4 u32 \
    match ip src 11.209.0.10/32 \
    police rate 1mbit burst 10k drop flowid :1

tc filter add dev eth0 parent ffff: \
    protocol ip prio 5 u32 \
    match ip src 11.209.0.20/32 \
    police rate 2mbit burst 10k drop flowid :2

tc qdisc add dev eth1 root handle 1: tbf rate 1.5Mbit burst 10k latency 20s
```

```
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 11.209.0.10/32 flowid 10:1
tc filter add dev eth1 parent 10:0 prio 2 protocol ip u32 match ip src 11.209.0.20/32 flowid 10:2
```

Listing 3.4: tc-egress-htb.sh

```
#!/bin/sh

tc qdisc del dev eth0 ingress
tc qdisc del dev eth1 root

tc qdisc add dev eth0 ingress handle ffff:

tc filter add dev eth0 parent ffff: \
    protocol ip prio 4 u32 \
    match ip src 11.209.0.10/32 \
    police rate 1mbit burst 10k drop flowid :1

tc filter add dev eth0 parent ffff: \
    protocol ip prio 5 u32 \
    match ip src 11.209.0.20/32 \
    police rate 2mbit burst 10k drop flowid :2

tc qdisc add dev eth1 root handle 1:0 htb

tc class add dev eth1 parent 1:0 classid 1:1 htb rate 1.2Mbit
# tc class add dev eth1 parent 1:1 classid 1:2 htb rate 700kbit ceil 700kbit
tc class add dev eth1 parent 1:1 classid 1:2 htb rate 700kbit ceil 1.2Mbit
# tc class add dev eth1 parent 1:1 classid 1:3 htb rate 500kbit ceil 500kbit
tc class add dev eth1 parent 1:1 classid 1:3 htb rate 500kbit ceil 1.2Mbit

tc filter add dev eth1 parent 1:0 protocol ip prio 1 u32 match ip src 11.209.0.10 flowid 1:2
tc filter add dev eth1 parent 1:0 protocol ip prio 1 u32 match ip src 11.209.0.20 flowid 1:3
```

Listing 3.5: script-r1.sh

```
#!/bin/sh

tc qdisc del dev eth0 ingress
tc qdisc del dev eth1 root

tc qdisc add dev eth0 ingress handle ffff:

tc filter add dev eth0 parent ffff: protocol ip prio 4 u32 match ip src 11.209.0.10/32 \
    match ip dst 16.209.0.40/32 police rate 1.2mbit burst 10k continue flowid :1
tc filter add dev eth0 parent ffff: protocol ip prio 5 u32 match ip src 11.209.0.10/32 \
    police rate 600kbit burst 10k drop flowid :2

tc filter add dev eth0 parent ffff: protocol ip prio 6 u32 match ip src 11.209.0.20/32 \
    match ip dst 16.209.0.50/32 police rate 300kbit burst 10k continue flowid :3
tc filter add dev eth0 parent ffff: protocol ip prio 7 u32 match ip src 11.209.0.20/32 \
    police rate 400kbit burst 10k drop flowid :4

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

tc class change dev eth1 classid 1:1 dsmark mask 0x3 value 0xb8
tc class change dev eth1 classid 1:2 dsmark mask 0x3 value 0x68
tc class change dev eth1 classid 1:3 dsmark mask 0x3 value 0x48
tc class change dev eth1 classid 1:4 dsmark mask 0x3 value 0x28

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

Listing 3.6: script-r2.sh

```
#!/bin/sh

tc qdisc del dev eth0 ingress
tc qdisc del dev eth1 root

tc qdisc add dev eth0 ingress handle ffff:

tc filter add dev eth0 parent ffff: protocol ip prio 4 u32 match ip src 12.209.0.30/32 \
    match ip dst 16.209.0.60/32 police rate 400kbit burst 10k continue flowid :5
tc filter add dev eth0 parent ffff: protocol ip prio 5 u32 match ip src 12.209.0.30/32 \
    match ip dst 16.209.0.60/32 police rate 300kbit burst 10k continue flowid :6
```

```

tc filter add dev eth0 parent ffff: protocol ip prio 6 u32 match ip src 12.209.0.30/32 \
    police rate 100kbit burst 10k drop flowid :7

tc qdisc add dev eth1 root handle 1:0 dsmark indices 4

tc class change dev eth1 classid 1:1 dsmark mask 0x3 value 0x68
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 0x28

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

```

Listing 3.7: script-r3.sh

```

#!/bin/sh

tc qdisc del dev eth2 root

tc qdisc add dev eth2 root handle 1:0 dsmark indices 8 set_tc_index

tc filter add dev eth2 parent 1:0 protocol ip prio 1 tcindex mask 0xfc shift 2

tc qdisc add dev eth2 parent 1:0 handle 2:0 htb

tc class add dev eth2 parent 2:0 classid 2:1 htb rate 2.4Mbit
# tc class add dev eth2 parent 2:1 classid 2:10 htb rate 1Mbit ceil 1Mbit
# tc class add dev eth2 parent 2:1 classid 2:20 htb rate 500kbit ceil 500kbit
# tc class add dev eth2 parent 2:1 classid 2:30 htb rate 400kbit ceil 400kbit
# tc class add dev eth2 parent 2:1 classid 2:40 htb rate 200kbit ceil 200kbit

# Nuevas modificaciones
tc class add dev eth2 parent 2:1 classid 2:10 htb rate 1Mbit ceil 2.4Mbit
tc class add dev eth2 parent 2:1 classid 2:20 htb rate 500kbit ceil 2.4Mbit
tc class add dev eth2 parent 2:1 classid 2:30 htb rate 400kbit ceil 2.4Mbit
tc class add dev eth2 parent 2:1 classid 2:40 htb rate 200kbit ceil 2.4Mbit

tc filter add dev eth2 parent 2:0 protocol ip prio 1 handle 0x2e tcindex classid 2:10
tc filter add dev eth2 parent 2:0 protocol ip prio 1 handle 0x1a tcindex classid 2:20
tc filter add dev eth2 parent 2:0 protocol ip prio 1 handle 0x12 tcindex classid 2:30
tc filter add dev eth2 parent 2:0 protocol ip prio 1 handle 0x0a tcindex classid 2:40

```