Modelo extendido de QoS sobre IPv6

Andres Barbieri
Director: Ing. Luis Marrone, Asesor Científico: Mg. Matías Robles
Fac. de Informática
Universidad Nacional de La Plata

Tesis enviada para el grado de
Magister en Redes de Datos

La Plata, Noviembre de 2015

Este trabajo esta dedicado a
mi amada y siempre compañera, Nancy

Agradecimientos

Agradezco al Ing. Luis Marrone por haberme guiado en el desarrollo de esta tesis, al Dr. Fernando Tinetti por haberme enseñado tantas cosas que se aplican en el transcurso de la misma, al Lic. Miguel Luengo por todos los conocimientos y oportunidades ofrecidas que me permitieron obtener experiencia en el área de redes de computadores, al Mg. Matías Robles y al Lic. Nicolás Macia por las conversaciones y recomendaciones que me ayudaron terminar el presente trabajo; ... a la Facultad de Informática y al Centro de Cómputo de la Universidad de La Plata (CeSPI) por haberme brindado el espacio y la formación.

Resúmen

En este trabajo se estudian y analizan los modelos actuales de QoS en redes IP, como se aplican a IPv4 y en particular a IPv6 [HD98]. Se analizan varias propuestas que ofrecen diferentes métodos de QoS sobre ambos protocolos. Se proponen modelos y se realizan pruebas sobre una implementación de referencia, testing, con un esquema de **QoS sobre IPv6**. El esquema que sigue la implementación de referencia se basa en la utilización del campo del datagrama IPv6: **Flow Label** (Etiqueta de flujo), del cual IPv4 carece. Se pretende mostrar que un modelo de QoS sobre un sistema operativo real, haciendo uso del campo Flow Label, es posible y la utilización del mismo para el manejo de tráfico puede ser provechosa.

keywords: Packet-switched Networking, QoS, IPv6, Flow Label, Traffic Class.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación y Objetivo General

El protocolo IPv6 propone mejoras importantes sobre su antecesor IPv4 tal cual fue concebido en los principios de los 70s. La principal reforma es el aumento del espacio de direcciones. Otro adelanto es la inclusión del manejo de la Calidad de Servicio (QoS), el cual, IPv4 previamente en su evolución agrega. Esta característica y una de las formas de implementarla es incorporada desde el inicio en el protocolo IPv6, quien mantiene el modelo de DiffServ [BBC+98] mediante el campo de DSCP (Differentiated services code point), llamado en el nuevo protocolo **Traffic Class**. El modelo DiffServ fue adicionado a IPv4 y luego incluido en IPv6 en su desarrollo como draft (borrador). Este esquema con un campo de 8 bits, de los cuales se utilizan 6, puede no ser suficiente para trabajar con gran cantidad de flujos de extremo a extremo de la red, no logrando brindar el tratamiento requerido. Por esta razón se busca mostrar un modelo alternativo de QoS para IPv6 haciendo un desarrollo de las componentes esenciales con una implementación de referencia.

Las redes de datos hoy se utilizan como transporte de información de diferentes clases de tráfico, desde la navegación utilizando la WWW hasta flujos de datos multimediales con requerimientos de tiempo real, por ejemplo llevando vídeo y/o voz. La integración de diferentes servicios sobre la misma infraestructura ha dado lugar a los términos TriplePlay y CuadruplePlay, donde se llega a la convergencia de las redes de datos que deben trasladar los distintos tipos de tráficos, dando como resultado que la QoS sea un requerimiento casi indispensable.

De acuerdo a los documentos de la IETF, RFC-6294 [HB11] y RFC-6436 [AJC11] el campo Flow Label no es utilizado en la práctica. Podríamos decir que existen varias

propuestas, pero ninguna se ha llegado a implementar para el uso en el protocolo. Debido a la falta de consenso, la incertidumbre generada por diferencias en su interpretación y la necesidad de tener implementaciones de IPv6 en termino, el campo fue ignorado dando como resultado que en la mayoría de los mensaje IPv6 va configurado a 0 (cero).

Recientemente se ha generado una propuesta para usarlo en el balanceo de carga con Equal Cost Multipath Routing (ECMP) y Link Aggregation (LAGs), la cual parece haber tenido mayor consenso, RFC-6438 [AC11]. Hasta este RFC, el Flow Label seguía teniendo el aspecto un campo experimental. En el documento de la IETF RFC-6437 [AC11], donde se definen las "nuevas" reglas de uso del campo, si bien este reemplaza a RFC-3697 [RCCD04] relajando varias normas que hacían bastante difícil su utilización sin romperlas, la aplicabilidad para marcar QoS parece ser descartada.

A lo largo de esta tesis se examinan algunos de los enfoques sobre el uso del campo para QoS. En la misma se vuelve a la posibilidad de marcar o poder determinar la QoS requerida en el campo del datagrama contradiciendo de alguna manera algunas recomendaciones del IETF. Se busca demostrar que propuestas como: [Pra04], [Rob08], [LK04], [BSM02], algunas no implementadas o solo desarrolladas en simuladores se pueden aplicar en sistemas reales. Se presenta otra variante para dar QoS diferenciando algoritmos de control de congestión, ofreciendo así dos enfoques para un manejo del tráfico usando el campo. Como resultado se obtienen ejemplos de uso del Flow Label para seleccionar diferentes posibles tratamientos por parte de la red logrando QoS.

Con respecto a la inclusión de este campo en la selección del tratamiento de QoS se han propuesto diferentes métodos para el mismo. Este texto dedica un capítulo en particular al estudio de las diversas alternativas mostradas por varias líneas de investigación.

El objetivo general de este trabajo es estudiar los modelos QoS que se ofrecen actualmente, posibles implementaciones y finalmente ver como estos se pueden extender a partir de las diferencias del protocolo IPv6 sobre IPv4 en un sistema operativo real.

1.2. Esquema del Trabajo

La primera parte del trabajo es esta introducción.

En el segundo capítulo se puntualizan las características generales del protocolo IPv6, no detallando en demasía y poniendo mayor énfasis en la estructura de los mensajes y el funcionamiento general. Para completitud se agrega un apéndice donde se muestran casos particulares del uso del protocolo. Los lectores conocedores de este tema pueden saltear este capítulo.

En el tercero se definen los principios de la QoS y se analiza el primer esfuerzo en agregarle esta funcionalidad a IP mediante el modelo IntServ [BBC+98]. Se analizan las componentes de la arquitectura, agregando un ejemplo de implementación en el apéndice correspondiente.

En el cuarto capítulo se detallan diferentes mecanismos de manejo de colas y elementos generales de QoS con su apéndice correspondiente incluyendo en este varios ejemplos.

En el quinto se detalla el modelo de DiffServ [NBBB98], se presentan componentes de la arquitectura y un ejemplo de implementación combinado con diferentes mecanismos de encolado, tratados en el apéndice del capítulo anterior.

En el sexto capítulo se hace una revisión de artículos, propuestas de RFCs y RFCs mismas que tratan enfoques para aprovechar el campo Flow Label hasta hoy en día. Parte de lo analizado sirve como base para continuar la tesis y mostrar la implementación desarrollada.

En el séptimo capítulo se muestran las herramientas para la implementación que existen en algunos sistemas operativos reales para poder trabajar con el campo Flow Label y sobre las cuales basar los tests. Se incluyen las herramientas utilizadas en los programas desarrollados para las pruebas.

En la octava parte se presenta las propuestas y pruebas específicas de como utilizar el campo como medio para brindar y marcar QoS sobre IPv6. En particular, de acuerdo a lo previamente indicado, se presentan dos enfoques:

- Primero se implementa un modelo similar a los presentados por otros autores como es el marcado de los parámetros requeridos para la QoS en el campo de IPv6.
- El segundo es un diseño para balanceo entre sesiones TCP con diferentes algoritmos de control congestión por detrás.

En el final de este capítulo se analizan los resultados obtenidos.

Finalmente en el noveno y último capítulo se resumen los resultados y se indican las conclusiones.

Según la introducción realizada previamente del contenido de cada capítulo, el texto está acompañado de varios apéndices, 4 (cuatro) en total, donde se muestran en la práctica varios de los conceptos analizados. En general parece suceder que en gran parte de textos donde se describe e investiga la QoS se encuentra un gran número de referencias y explicaciones de como se utilizaría, pero faltan ejemplos ilustrativos aplicados. Como objetivo secundario de este texto se trata de poder ejemplificar con casos prácticos posibles escenarios de aplicación de QoS en redes de datos.

Capítulo 2

Estudio general del protocolo IPv6

En este capítulo se realiza un breve repaso de las características principales y el funcionamiento del protocolo IPv6, el cuál esta definido por la IETF en el documento RFC-2460 [HD98]. El punto en el cual se coloca mayor énfasis estará en la estructura del mensaje y algunas otras cuestiones que lo diferencian de IPv4. Conocer la estructura del datagrama es importante porque al momento de clasificar el tráfico para brindar una tratamiento diferente se debe saber que valores comparar para distinguir los flujos.

Para profundizar en el tema con cuestiones como tunneling, mecanismos de transición, ruteo dinámico, multicast, encabezados de extensión, seguridad y otras tantas, se puede consultar en la numerosa documentación y bibliografía que analiza y explica como poner en marcha a IPv6: [Los03], [iJi04], [Hag06], [Sto06], [PLAG06], [MSSH11]. En el anexo del capítulo, anexo "A", se ilustra con algunos casos de usos específicos utilizando ICMPv6.

2.1. Direcciones IPv6

El principal impulso para IPv6 fue, y es, el aumento del espacio de direcciones. A forma introductoria, se realiza un repaso del esquema de direcciones aportado por este protocolo. Los detalles de las direcciones son necesarios al momento de distinguir los flujos.

2.1.1. Notación de direcciones

Las direcciones tienen una longitud de 128 bits y están representadas en grupos de 16 bits escritos en valores hexadecimales y separados por un dos puntos ":" (colon).

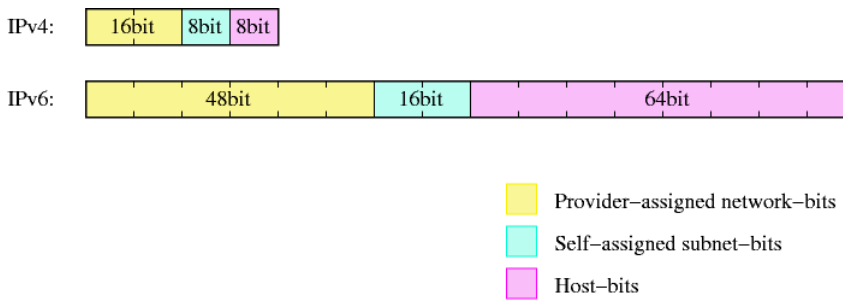


Figura 2.1: Esquema de direcciones IPv6 con subredes

Puede utilizarse la notación comprimida de doble dos puntos ("::") para representar grupos de ceros consecutivos, pero sólo puede usarse una vez en la expresión (en la secuencia más larga de ceros consecutivos). Dentro de cada grupo, los ceros más a la izquierda se pueden suprimir. Se define un formato propuesto de 64 bits para la parte de red y 64 para el host.

Las subredes no son expresadas utilizando la máscara de subred sino la notación longitud de prefijo: `Dirección/Prefijo`, la misma usada para CIDR (Classless Inter-Domain Routing) en IPv4. El formato de 64 y 64 indicado anteriormente no es estricto y puede ser modificado permitiendo redes con mayor o menor cantidad de hosts. Una recomendación para el caso de direcciones unicast globales es utilizar 32 bits para la parte de red, permitiendo subredes de 32 bits dentro de la red principal o 48 bits para la red, permitiendo subredes de 16 bits. Hasta 2011 el bloque mínimo asignable era un /48. En la figura 2.1 tomada de la documentación de la Guía de NetBSD se compara dos posibles esquemas de sub-netting entre IPv4 e IPv6 para direcciones unicast.

2.1.2. Alcances y tipos de direcciones IPv6

Existen diferentes grupos o tipos de direcciones dentro del rango de 128 bits. Cada grupo tiene un alcance, llamado en inglés Address Scope. Los alcances están definidos en RFC-4007 [DHJ+04]. Los nombres de las clases se especifican en el idioma nativo en el cual fueron definidas.

Host Scope/Loopback: sólo tienen sentido en el host local. Nunca deben ser ruteadas ni aparecer en la red. Se utilizan cuando un host se envía datos a si mismo. Se puede representar de varias maneras. Ejemplo: `::1/128`, `::1`, `0:0:0:0:0:0:0:1`. Definida primero en RFC-2373 y RFC-3513 y finalmente RFC-4291[HD06]. Usada para localhost, es unicast.

Unspecified: Sin especificar. Ejemplo: `::/128`, `00...0` (128 bits), `0:...:0:0`. Definido en RFC-4291. No se debe asignar a ningún nodo. Indica la ausencia de dirección y es utilizada en el proceso de descubrimiento de direcciones o en la especificación de ANY. Igual a `0.0.0.0` en IPv4. No tienen alcance y no se asignan como destino en ningún paquete.

Link Scope: Alcance de enlace, red local. Son unicast, direcciones válidas únicamente en el dominio de broadcast. Sólo pueden utilizarse para direccionar nodos de una red local. Los routers no deben reenviar paquetes con estas direcciones. Usadas por ejemplo para:

- Descubrimiento de los vecinos, como el router de la red.
- Obtención automática de una dirección.
- Poder trabajar de forma local aún sin la presencia de un router.
- Detección de direcciones duplicadas.

Prefijo/Rango: `FE80::/10, FE80:: .. FEBF:...:FFFF`.

Site Scope: privadas/locales, similares a 192.168.0.0/16, 10.0.0.0/8 ó 172.16.0.0/12 en IPv4. Definidas inicialmente en RFC-2373 y luego, en RFC-3513. Los routers no deberían hacer forwarding de paquetes IPv6 con estas direcciones fuera del sitio.
Prefijo/Rango: `FEC0:/10, FEC0:: .. FEFF:...:FFFF`.
La RFC-3879 [HC04] las dejó fuera de uso debido a su ambigüedad y a la confusa definición del término "site". Indica, además, que el direccionamiento privado crea problemas generando confusión. Propone que sean re-asignadas para otros usos. Eran unicast.

Local Unique: Locales, únicas y unicast. Son direcciones que reemplazan a las direcciones Site Scope. Definidas en RFC-4193[HH05]. Parte de la dirección se crea con un generador pseudo-aleatorio. Estas direcciones no deben ser ruteadas fuera del sitio.
Ejemplos de prefijos: `FCxx, FDxx, ....`

Global Scope: públicas, globales y unicast, como por ejemplo 163.10.0.0/16 en el caso de IPv4. Definidas en RFC-4291. No definen un rango particular, sino que abarcan todas las direcciones que no están cubiertas por los otros alcances. Prefijo/rango: `2000/3, 2000:: .. 3FFF:...:FFFF`.

Compatibility Scope: Compatibilidad con IPv4. Utilizadas cuando se encapsula IPv4 dentro de IPv6. Son unicast.

Multicast: Multi-difusión. En IPv6 no existen más las direcciones de broadcast. Para este propósito se deben utilizar direcciones de Multicast.
Prefijo/Rango: `FF00::/8, FF00:: .. FFFF:...:FFFF`.

Anycast: son direcciones especiales que indican un destino entre muchos posibles. Pueden ser utilizadas para cuestiones de balanceo de carga, por ejemplo, para consultar el DNS más cercano, o cualquiera dentro de un grupo de varios DNS. Las direcciones son tomadas del grupo Unicast Global o Site-Scope. De acuerdo a RFC-2373 nunca deberían ser utilizadas como origen de un paquete.

A continuación se muestran algunos ejemplos de tipo de direcciones.

2.1.2.1. Loopback Address (::1/128)

Usada para la interfaz loopback `lo/lo0` (localhost), similar a la dirección `127.0.0.1` de IPv4.

```
# ifconfig lo
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          ...

# ping6 -c2 ::1
PING ::1(::1) 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.047 ms
64 bytes from ::1: icmp_seq=2 ttl=64 time=0.070 ms

--- ::1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.047/0.058/0.070/0.013 ms

# ping6 -c2 0::1
PING 0::1(::1) 56 data bytes
64 bytes from ::1: icmp_seq=1 ttl=64 time=0.051 ms
64 bytes from ::1: icmp_seq=2 ttl=64 time=0.065 ms

--- 0::1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.051/0.058/0.065/0.007 ms

# grep localhost /etc/hosts
127.0.0.1 localhost
::1             ip6-localhost ip6-loopback

# ping6 -c2 ip6-localhost
PING ip6-localhost(ip6-localhost) 56 data bytes
64 bytes from ip6-localhost: icmp_seq=1 ttl=64 time=0.047 ms
64 bytes from ip6-localhost: icmp_seq=2 ttl=64 time=0.065 ms

--- ip6-localhost ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.047/0.056/0.065/0.009 ms
```

2.1.2.2. Sin especificar, Any Address (::/0)

No tienen alcance y no se asignan como direcciones destino.
Ejemplo en la tabla de conexiones o como default gateway:

```
# netstat -atn | grep tcp6
tcp6       0      0 :::22         :::*        LISTEN

# netstat -6 -nr | grep "::/0"
::/0       2001:db8:0:f101::1         UG   1024 0     0 eth0
::/0       ::                         !n   -1   1   630 lo
```

2.1.2.3. Direcciones de enlace local (Link-local)

Usadas en la auto-configuración de direcciones. Siempre se encuentran configuradas en una interfaz con IPv6 habilitado. De acuerdo a RFC-4291 un nodo no podría utilizar IPv6 si no tiene una dirección de este tipo asignada. Son locales y unicast. Algunos ejemplos de los posibles prefijos de 16 bits que pueden tener se listan a continuación.

```
FE80: # El único prefijo utilizada hasta el año 2011
FE81:
...
FEBF:
```

Ejemplo:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:21:CB:97:92
          inet addr:172.20.1.100  Bcast:172.20.1.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:21ff:fecb:9792/64 Scope:Link
          ...
```

Las direcciones de enlace se basan en el ID EUI-64. El identificador de interfaz único IEEE EUI-64 (Extended Unique Identifier) es derivado a partir de la dirección Ethernet MAC con formato EUI-48. En la figura 2.2 se muestra el proceso de generación automática de una dirección IPv6 de link-local a partir a de una dirección MAC.

El valor `0xFFFE` es utilizado, según la IEEE, debido a que es reservado y los fabricantes no podrían incluirlo en un valor real EUI-64. Será solo usado a partir de direcciones EUI-48 (MAC). Para direcciones EUI-64, directamente se usa el valor dado. El otro paso consiste en invertir el bit Universal/local (UL) en la parte del OUI (Identificador del fabricante). En el caso de la MAC, “0” significa global y “1” local. La necesidad de la inversión de este bit parece residir en otorgar a los administradores la facilidad de configurar direcciones a mano más fácilmente sin colocar este bit en “1” en el caso de no tener un ID por hardware. El identificador de interfaz de 64 bits se puede utilizar en todas las direcciones de tipo unicast que tengan un prefijo de 64 bits. Un prefijo de 64 bits más un identificador, también, de 64 bits conforman una dirección IPv6 de 128 bits.

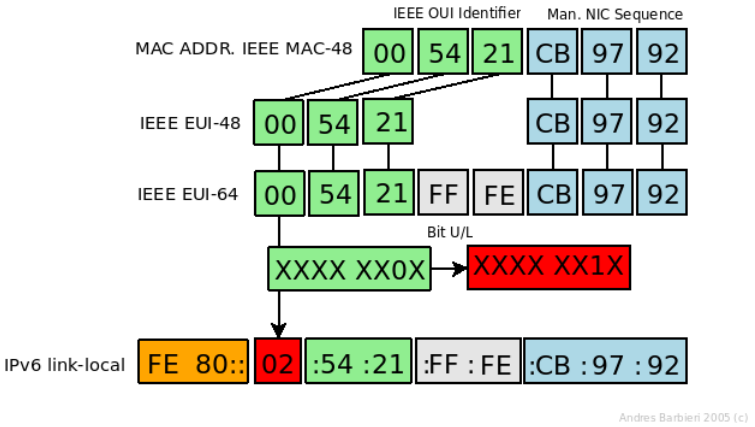


Figura 2.2: Generación de una dirección Link-Local con EUI-64

2.1.2.4. Direcciones locales privadas de sitio (Site-local)

Utilizadas para direcciones de intranets. Están definidas en los primeros RFCs que describen el protocolo, en la actualidad están desaconsejadas (como se explicó anteriormente). A continuación se muestran ejemplos:

```
# ifconfig eth0 inet6 add fec2::0100/64

# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:21:CB:97:92
          inet addr:172.20.1.100  Bcast:172.20.1.255  Mask:255.255.255.0
          inet6 addr: fec2::100/64 Scope:Site
          inet6 addr: fec1::100/64 Scope:Site
          inet6 addr: fec0::100/64 Scope:Site
          inet6 addr: fe80::5054:21ff:fecb:9792/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          ...
```

2.1.2.5. Direcciones locales únicas (Unique Local)

El prefijo que define estas direcciones es FC00/7. El bit 8 puede tener el valor 1=Local ó 0=Reservado. Al tener 1, el prefijo se transforma en FD00/8. El siguiente campo de 40 bits, llamado Global ID, tiene un valor aleatorio. Esto nos brinda un prefijo de 48 bits que puede ser asignado a un sitio. Quedan 16 bits para crear subredes dentro del sitio y los 64 bits restantes pueden ser asignados a los host.

```
# python createLULA.py
Your 'Locally Assigned Global ID' is fd08:66e1:66f7::/48
Your 'Locally Assigned Global ID' is fdbc:189:6f98::/48
Your 'Locally Assigned Global ID' is fd94:ad66:9c30::/48
Your 'Locally Assigned Global ID' is fd8f:c4e6:75ff::/48
Your 'Locally Assigned Global ID' is fd2f:77f0:d7ce::/48
Your 'Locally Assigned Global ID' is fd46:3b37:ff9::/48
Your 'Locally Assigned Global ID' is fd7f:bce2:b0f1::/48
...
```

```
#  ifconfig eth0 inet6 add fd08:66e1:66f7::100/64

# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:21:CB:97:92
          inet addr:172.20.1.100  Bcast:172.20.1.255  Mask:255.255.255.0
          inet6 addr: fd08:66e1:66f7::100/64 Scope:Global
          inet6 addr: fec2::100/64 Scope:Site
          inet6 addr: fec1::100/64 Scope:Site
          inet6 addr: fec0::100/64 Scope:Site
          inet6 addr: fe80::5054:21ff:fecb:9792/64 Scope:Link
          ...
```

2.1.2.6. Direcciones globales unicast/basadas en provider

El término "Provider Based" ha quedado obsoleto y simplemente se las llama direcciones globales unicast. Algunos ejemplos de estas son:

```
2000:
2001:          Assigned by provider for hierarchical routing
2002:          6to4 addresses
....
3FFE:FFFF::/32 Reserved for examples
2001:0DB8::/32 Reserved for examples
3FFE:          6bone test, now free
3FFE:FFFF::/32 6bone Reserved for examples, now free
3FFF:
```

Por ejemplo la dirección: 2001:db8::/32 es usada con fines de ejemplos y documentación del protocolo. La dirección 2800:340::/32 es la dirección asignada a la UNLP por LACNIC, La dirección de red 2001:1318:A001::/64 era la que antiguamente asignaba la institución RETINA a la UNLP. 2800:110:FF:F8::2/64 es el actual prefijo otorgado por la ARIU a la UNLP. Estos detalles se pueden consultar vía los servicios de WHOIS.

2.1.2.7. Direcciones multicast y anycast

Las direcciones multicast en IPv6 comienza con el prefijo FF00/8. Los siguientes 4 bits (del 9 a 12) indican si la dirección es permanente (asignadas por el IANA) o temporal. Los otros 4 bits indican el alcance de la dirección (node, link, site, etc).

Existen direcciones multicast que deben estar asignadas a todos los nodos por ejemplo la primera `FF02::1`. Algunos ejemplos de dir de multicast son:

```
FF02::1  - Dirección multicast de todos los nodos  ALL-SYSTEMS.MCAST.NET
FF02::2  - Dirección multicast de todos los routers ALL-ROUTERS.MCAST.NET
...
FF02::5  - OSPFv6 AllSPFRouters ospf-all.mcast.net
FF02::6  - OSPVv6 AllDRouters   ospf-dsig.mcast.net (En caso de DR/BDR)
FF02::9  - RIPv2-ng rip2-routers.mcast.net (RIPv2-ng)
FF02::A  - EIGRP igrp-routers.mcast.net (cisco)
```

Para cada dirección unicast IPv6 existente en un nodo se debe generar una dirección multicast de nodo solicitado. Estas direcciones tiene el prefijo FF02::1/104. Los últimos 24 bits son una copia de los últimos 24 bits de la dirección unicast que representan. Estas direcciones son utilizadas por el protocolo Neighbor Discovery (ND). En este caso son solo de alcance local.

Para multicast de alcance global se tiene por ejemplo las direcciones IPv6 usadas como anycast para los root DNS:

```
c.root-servers.net. 604795 IN AAAA 2001:500:2::c
k.root-servers.net. 604751 IN AAAA 2001:7fd::1
l.root-servers.net. 604761 IN AAAA 2001:500:3::42
j.root-servers.net. 604756 IN AAAA 2001:503:c27::2:30
```

Existen también direcciones anycast conocidas como "Subnet Device Anycast Addresses". Estas direcciones es el valor del prefijo de la interfaz seguida por una secuencia de ceros (donde estaría el interfaz ID). Esta dirección puede ser usada para alcanzar al dispositivo en el link donde esta conectado solamente, a la subred. Todas las interfaces de un router que están unidas a una subred tienen esta dirección configurada para esa subred.

Por ejemplo en el caso de un router cisco se podría configurar como:

```
Router(config-if)# ipv6 address 2002:db8:1234::/128 anycast
```

2.1.2.8. IPv4 mapeado en IPv6 (0:0:0:0:0:ffff:a.b.c.d/96)

Las direcciones IPv4-mapped-IPv6, ofrecen un mapeo de IPv4 a IPv6 y son utilizadas generalmente en sockets creados por servicios con soporte de IPv6 pero "bindeados" a una dirección IPv4. Su alcance es global. Usados para sistemas híbridos con soporte de dual-stack.

```
0:0:0:0:0:FFFF:X.Y.Z.W
::FFFF:X.Y.Z.W
0:0:0:0:0:FFFF:192.168.0.1
::FFFF:192.168.0.1
0:0:0:0:0:FFFF:C0A8:0001
```

Ejemplo:

```
# ifconfig eth0 inet6 add ::FFFF:192.168.0.1
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:21:CB:97:92
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::200:21ff:fecb:9792/64 Scope:Link
          inet6 addr: ::ffff:192.168.0.1/0 Scope:Global
          ...
```

2.1.2.9. IPv4 compatible IPv6 (0::a.b.c.d/96)

Sistema de IPv4 mapeado a IPv6. Utilizadas para compatibilidad en sistemas dual-stack, objetivo similar al anterior. Formato deprecado por RFC-4291. Alcance compatibilidad con IPv4.

```
0000:0000:0000:0000:0000:0000:X.Y.Z.W
::X.Y.Z.W
0:0:0:0:0:0:192.168.0.1
::192.168.0.1
0:0:0:0:0:0:C0A8:0001
```

Ejemplo:

```
# ifconfig eth0 inet6 add ::192.168.0.1
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:00:21:CB:97:92
          inet addr:192.168.0.1  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::200:21ff:fecb:9792/64 Scope:Link
          inet6 addr: ::192.168.0.1/0 Scope:Compat
          ...
```

2.1.3. Tiempo de vida de las direcciones IPv6

Las direcciones en IPv6 son asignadas a las interfaces de los dispositivos por un período de tiempo. Este puede ser infinito o determinado por un valor. Una dirección puede estar en uno de los siguientes momentos de acuerdo a su tiempo de vida:

Tentative: Tentativo. La dirección está en el proceso de verificación de su unicidad. Un nodo no debe aceptar paquetes que tienen como dirección destino una dirección en este estado.

Valid: Válido. Se le ha probado su unicidad a la dirección. Este estado cubre los estados preferido y desaconsejado.

Prefered: Preferido. Indica el período de tiempo en el que una dirección puede ser usada en forma segura para enviar y recibir tráfico.

Deprecated: Desaconsejado. El tiempo de vida preferido ha expirado pero la dirección todavía es válida. No se aconseja su uso para establecer nuevas comunicaciones, pero las existentes pueden continuar con este valor.

Invalid: Inválido. El tiempo de vida válido expiró y no se puede enviar ni recibir datos utilizando esa dirección.

En la figura 2.3 se ilustran los estados.

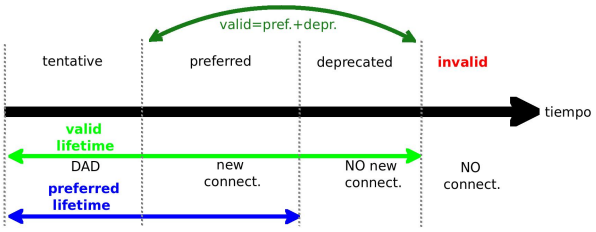


Figura 2.3: Ciclo de posibles estados en el tiempo de vida de una dir. IPv6

2.2. Estructura del datagrama IPv6

Es importante notar que la cabecera IPv6 se ha simplificado notablemente con respecto a la cabecera IPv4. Las funcionalidades que no se encuentran en este encabezado pueden lograrse mediante headers (cabeceras) de extensión. Con los campos bases, IPv6 ofrece las siguientes funcionalidades básicas, que se pueden derivar de ver los campos de la figura 2.4:

- Direccionamiento (Addressing).
- Ruteo/Switching/Forwarding (Routing).
- Multiplexado/Encapsulado de los protocolos superiores (Multiplexing/Encapsulation).
- Marcado de tráfico para QoS.
- Extensión de funcionalidad mediante cabeceras de extensión.

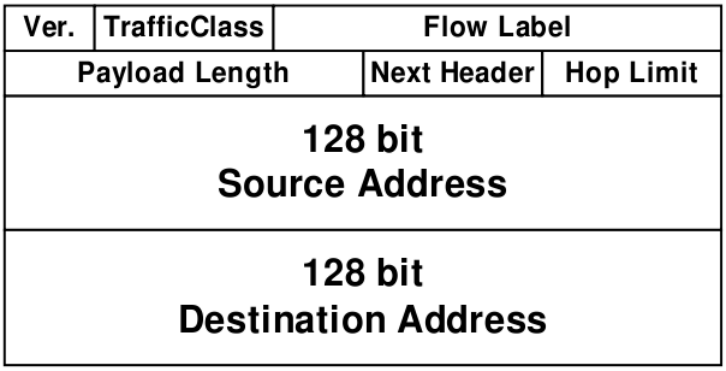


Figura 2.4: Datagrama IPv6

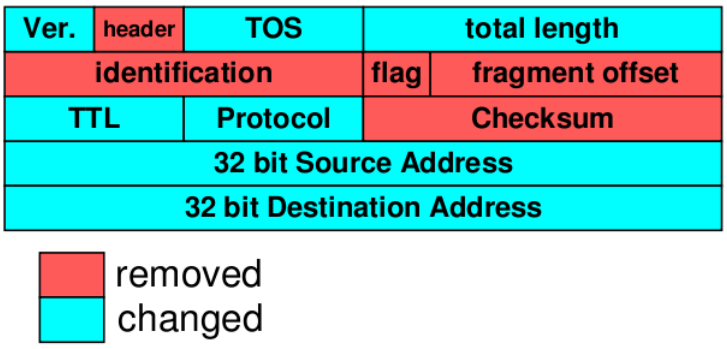


Figura 2.5: Campos del datagrama IPv4 que cambian o se eliminan

En la figura 2.5 se muestran los cambios realizados sobre la cabecera IPv4 para generar una de tipo IPv6. Las opciones de IPv4 también son removidas pero no se muestran en el gráfico.

Notar que se elimina el checksum del encabezado, siendo ahora obligatorio para los protocolos incluidos en el datagrama IPv6 y la fragmentación se agrega como una extensión con el objetivo de que no sea necesaria. Puede evitarse esta última mediante un procedimiento de descubrimiento del tamaño mínimo del MTU en el camino, o implementada solo en los extremos. La cabecera IPv6 al eliminar las opciones queda de tamaño fijo: 40 bytes. Las opciones se manejan como extensiones que no necesariamente se deben analizar hop-by-hop. Esta característica le da la posibilidad de genera algoritmos de switching más eficientes.

2.2.1. Campos del encabezado IPv6

2.2.1.1. Campo versión

De la misma forma que en IPv4, IPv6 tiene un campo de 4 bits para indicar la versión del protocolo. En este caso, se usa el valor 6 en lugar de 4. No se utilizó el valor 5 porque ya había sido asignado a otro protocolo, Internet Stream Protocol (ST), definido en el RFC-1819 ST-II. El mismo también se encuentra en la tabla de protocolos.

```
# grep ST /etc/protocols
st 5 ST # ST datagram mode
```

Es importante hacer notar que no son dos versiones de un mismo protocolo, son protocolos de red distintos que cumplen funcionalidades similares.

2.2.1.2. Campo clase de tráfico

Además del campo **Version**, se tienen entre estos el de **(TC) TrafficClass**, equivalente al **ToS/DSCP** de IPv4. Este campo es de 8 bits y a menudo también es llamado **Prioridad**, justamente porque permite identificar entre diferentes clases o prioridades de paquetes. Al momento de la definición del documento también se desarrollaban los relacionados con el modelo DiffServ (Modelo de Servicios Diferenciados) sin necesidad de hacer un set-up de flujo explícito. La forma de interpretarlo y configurarlo para el modelo DiffServ está regida en los documentos RFC-2474[NBBB98] y RFC-2475 [BBC+98], y es la misma tanto para IPv4 como para IPv6.

Los siguientes requerimientos se aplican a este campo:

- La interfaz que ofrece la capa de red IPv6 a las capas superiores debe brindar un medio para que estas indiquen el valor de los bits de este campo. En el caso configurado a 0 (cero) significa default traffic class, tratamiento default como "best-effort".

- Los nodos que soporten un uso específico, experimental o estándar, para este campo están permitidos a cambiarlo, transmitirlo y/o recibirlo de acuerdo a el uso que se le quiera dar. Si los nodos no lo soportan deberán ignorarlo y dejarlo sin cambios.

- Las capas superiores deben asumir que el valor del campo recibido es el mismo que fue transmitido por el origen, aunque habitualmente esto no sucede.

2.2.1.3. Campo etiqueta de flujo (Flow Label)

El campo **Flow Label** de 20 bits, especificado en la RFC-2460, es usado por los nodos para etiquetar/marcar paquetes que pertenezcan a un mismo flujo. Inicialmente tuvo otro tamaño, 24 bits, pero finalmente el estándar lo estableció en 20 bits. Los flujos o paquetes marcados con un valor de 0 (cero) indican que no tienen etiqueta. La clasificación deberá hacerse tomando en cuenta también las 2 direcciones origen y destino. Los paquetes serán procesados de acuerdo a flujos por los nodos que lo soporten en un modelo básicamente sin estado de acuerdo a la IETF. El tratamiento que recibirán de acuerdo al valor no es especificado. El uso de un modelo con estados tampoco es definido. El uso del campo y las características serán tratadas en particular en próximos capítulos. El marcado podrá ser de un nodo hacia otro o de extremo a

extremo, de esta forma se podrá recibir un tratamiento distinto de parte de la red. El modelo recomendado es el de extremo a extremo (end-to-end).

2.2.1.4. Tamaño de los datos/carga (Payload)

El campo **PayloadLength** indica la cantidad de bytes que lleva como datos a continuación del IPv6 header. El campo es de 16 bits. Cualquier cabecera de extensión será considerada como datos de carga y no como encabezado.

IPv6 permite la alternativa de trabajar con datagramas mayores a los 64KB limitados por los 16 bits de este campo. Para esto agrega un encabezado de extensión definido por RFC-2675 donde el campo para el valor del tamaño es de 32 bits, permitiendo longitudes de hasta 4GB. Estos datagramas reciben el nombre de Jumbograms o Jumbogramas. Es importante notar que los protocolos de transporte podrían llegar a requerir modificaciones para soportar o aprovechar esta extensión, como es el caso de TCP con el MSS o UDP con el campo longitud.

2.2.1.5. Próximo encabezado

El campo **NextHeader** cumple una función similar al campo **Protocol** de IPv4, pero permitiendo mayor flexibilidad. A partir de este campo se indica que contenido lleva, por ejemplo: TCP, UDP, ICMPv6, IPv4 o extensiones propias de IPv6. Los valores básicos considerados son los mismos que para IPv4, indicados por RFC-1700 en la sección "Assigned Internet Protocol Numbers". Para el caso de las extensiones se logra mediante encabezados de extensión al protocolo, que permiten dar una forma de encapsulamiento de las opciones dentro del datagrama. Algunas de las funcionalidades que brinda una implementación IPv6 a través de estos encabezados son:

- Opciones Hop-by-Hop.
- Routing (Type 0).
- Fragmentación.
- Opciones para destino.
- Autenticación y cifrado dando seguridad al payload y a parte del encabezado (IPSec).
- Mobile IP (IP móvil).

Se recomienda por IETF que los encabezados de extensión aparezcan en un orden establecido:

1. IPv6 header.
2. Hop-by-Hop Options header.
3. Destination Options header.
4. Routing header.
5. Fragment header.

Un nodo que recibe un paquete IPv6 debe procesar las cabeceras de extensión en el orden en el que se encuentran. Está prohibido que un nodo recorra las cabeceras buscando una en particular. Como ejemplo se puede mencionar que la cabecera para soportar Jumbogramas es de Hop-by-Hop.

2.2.1.6. Límite de saltos

Luego se encuentra **Hop-Limit** que reemplaza al campo TTL (Time to Live) de IPv4. Para este último solo se cambia el nombre pero la semántica de implementación se conserva prácticamente igual: se decrementa salto a salto y si llega a 0 (cero) se descartará, salvo que sea el destino. El valor mayor sigue siendo 255 (8 bits) y el menor 0. Este campo puede utilizase para limitar el ámbito/alcance de los datagramas IPv6. En la práctica para IPv4 pueden encontrarse raras implementaciones que usan el campo para limitar el tiempo de vida. Esta semántica es completamente descartada en IPv6 y los protocolos que requieran la funcionalidad deberán implementarla de forma separada.

2.2.1.7. Direcciones

Por último tenemos las dos direcciones, origen y destino de 128 bits c/u. En la figura 2.7 se puede ver una captura de un mensaje ICMPv6 donde se muestran todos los encabezados fijos de IPv6, encapsulado en Ethernet II.

2.3. Encapsulamiento de IPv6

Para cada protocolo de nivel de enlace sobre el cual se monta IPv6, de la misma forma que sucede para IPv4, se define la forma de encapsularlo sobre la trama. Esto debe definirse de forma estándar para no generar incompatibilidades.

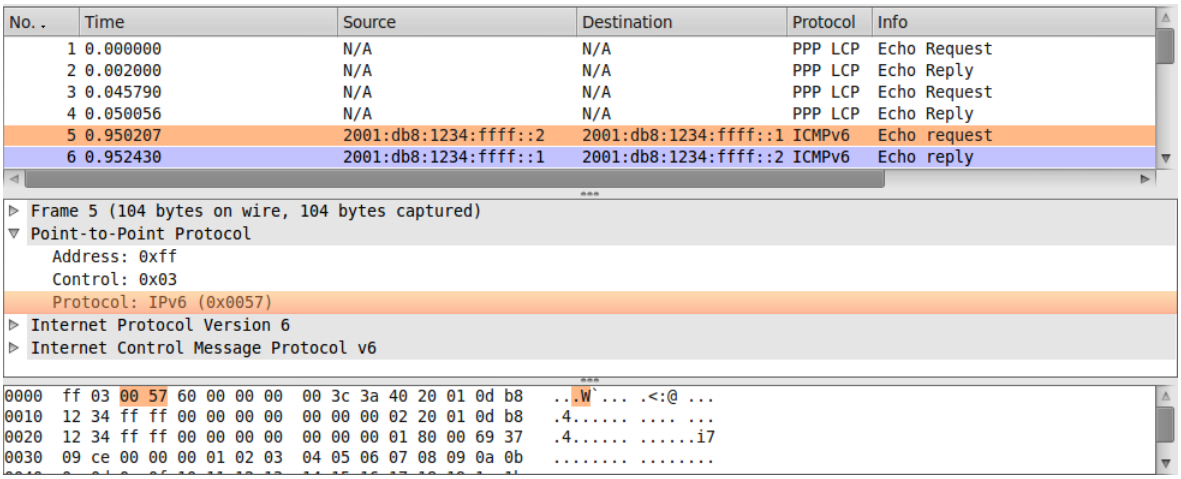


Figura 2.6: IPv6 encapsulado en PPP

2.3.1. Point-to-Point

Para el caso de SLIP/CSLIP (Compressed SLIP), IPv6 no tiene implementaciones que ejecuten sobre estos protocolos. Para conexiones punto a punto, el protocolo estándar más utilizado es PPP (Point-to-Point Protocol). Para este sí se define la forma de encapsulamiento mediante documentos estándares: RFC-2472[HA98]. En la figura de la captura 2.6 se presenta un ejemplo. En el campo protocolo el código para IPv6 es diferente al valor definido por IEEE debido a que no es un estándar de este grupo. Cisco además define su solución para cHDLC. También se define para otras tecnologías multi-punto no broadcast como ATM, Frame-Relay, etc.

2.3.2. Ethernet/IEEE 802

2.3.2.1. Ethernet II

Para redes LAN cableadas, el protocolo más utilizado, sin dudas, es Ethernet. El encapsulamiento para este se define en RFC-2464[Cra98]. Esta especifica un MTU máximo de 1500 bytes. Con respecto al MTU mínimo no se menciona nada ya que un paquete IPv6 tiene 40 bytes de cabecera y cualquier otra cabecera llenaría los 6 bytes restantes para los 46 bytes mínimos. El valor para el campo de Ethertype o Type directamente es de **0x86DD**. El estándar especifica también cuestiones como el mapeo de una MAC-48 a un identificador EUI-64 y la dirección de multi-difusión a nivel de enlace. En la captura mostrada en la figura 2.7 se puede observar un ejemplo de este encapsulamiento.

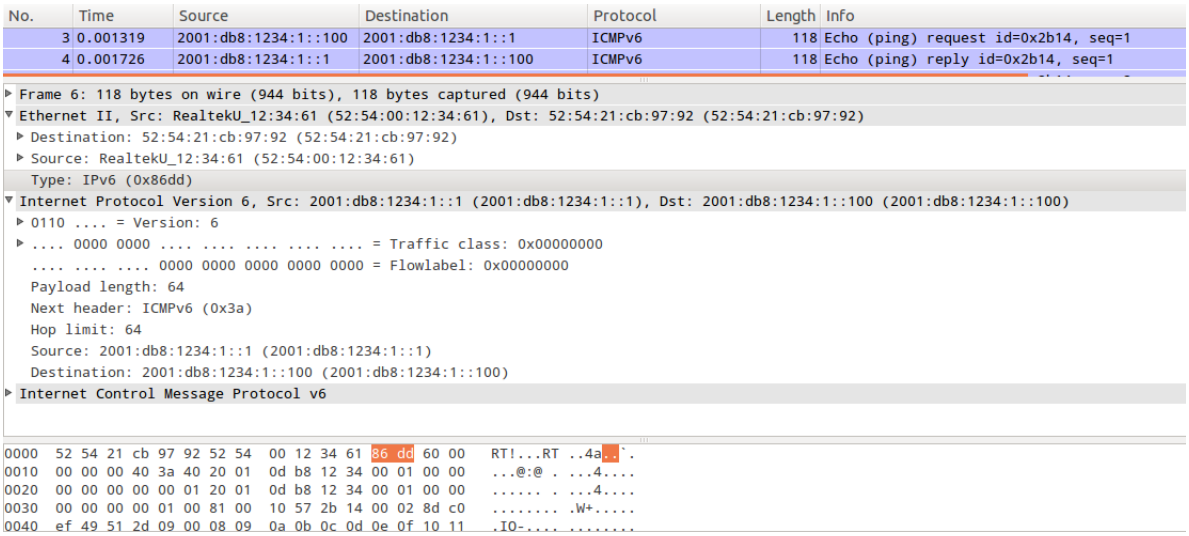


Figura 2.7: IPv6 encapsulado en Ethernet II

2.3.2.2. Redes wireless IEEE 802.11

A continuación se muestra la configuración de una interfaz Wifi que tiene habilitado el protocolo IPv6, en este caso solo se ve una dirección link-local.

```
# iwconfig wlan0
wlan0      IEEE 802.11bg  ESSID:"TESTIPV6"  Nickname:""
           Mode:Managed  Frequency:2.437 GHz  Access Point: 00:1C:10:CC:4F:6D
           Bit Rate=54 Mb/s   Tx-Power:32 dBm
           Retry min limit:7   RTS thr:off   Fragment thr:off
           Power Managementmode:All packets received
           Link Quality=5/5  Signal level=-32 dBm  Noise level=-78 dBm
           Rx invalid nwid:0  Rx invalid crypt:537  Rx invalid frag:0
           Tx excessive retries:0  Invalid misc:0   Missed beacon:0

# ifconfig wlan0
wlan0      Link encap:Ethernet  HWaddr 00:1f:e1:3c:f4:a4
           inet addr:10.168.1.160  Bcast:10.168.1.255  Mask:255.255.255.0
           inet6 addr: fe80::21f:e1ff:fe3c:f4a4/64 Scope:Link
           ...
```

Se observa que para IEEE 802.11 se utiliza un esquema de encapsulamiento análogo al usado para IEEE 802.3 con 802.2. En la figura 2.8 se presenta un ejemplo de una captura de una trama IEEE 802.11.

2.4. Internet Control Message Protocol v6 (ICMPv6)

ICMPv6 es la implementación de ICMP para IPv6. El mismo está definido en la RFC-4443 [CDG06]. ICMPv6, a diferencia de lo que parece suceder entre ICMP e IPv4, es parte integral del protocolo IPv6. ICMPv6 cubre la funcionalidad de ICMP:

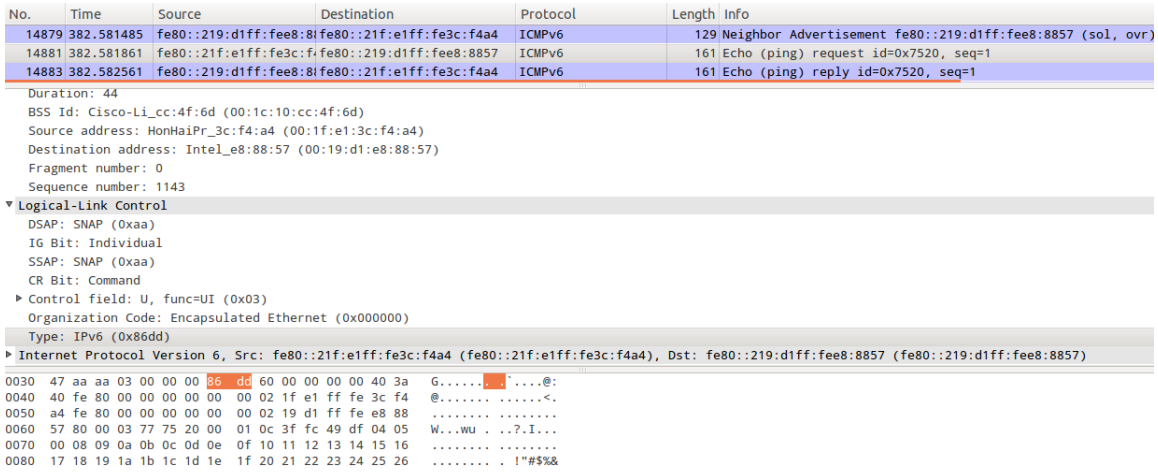


Figura 2.8: IPv6 encapsulado en IEEE 802.11

- Protocolo de diagnóstico.
- Reporte de mensajes de información, alerta y error.
- Redirecciones y cambio de rutas.

Pero además, en ICMPv6 se agregan la siguientes funcionalidades:

- Reemplazo de ARP con el proceso de Neighbor Discovery Protocol (NDP).
- Auto-configuración sin estado con Router Advertisement Protocol (RA).
- Reemplazo de IGMP con Multicast Listener Discovery (MLD).
- Mobile IPv6 con por ejemplo Home Agent Address Discovery.

En el siguiente texto tomado de la salida de un sistema se muestra la asignación de número de protocolo para ICMPv6.

```
# grep ipv6-icmp /etc/protocols
ipv6-icmp 58 IPv6-ICMP # ICMP for IPv6
```

2.5. Ejemplos con IPv6

Para finalizar el capítulo se lo invita al lector interesado en más detalles a consultar en el primer Apéndice, el “A”, dos ejemplos prácticos de aplicación de ICMPv6, el primero como reemplazo de ARP para el descubrimiento de vecinos y el segundo para la auto-configuración sin estado.

Capítulo 3

Introducción a QoS & ISA (Integrated Services)

3.1. Introducción

Se puede definir **QoS, calidad de servicio,** como el conjunto de tecnologías que permiten a la red administrar los recursos y manejar los efectos de la congestión de forma "óptima", o al menos mejor que el modelo "best-effort". Es la "habilidad" de la red de proveer mejor tratamiento, un servicio diferente de forma selectiva al tráfico que la cursa.

Según la ITU-T, de acuerdo al documento **E.800** de 2008, se indica **QoS:** como la totalidad de las características de un servicio de telecomunicaciones que determinan su capacidad para satisfacer las necesidades explícitas e implícitas del usuario del servicio.

El modelo ISA (Integrated Services Architecture) RFC-1633 [BCS94], en español, Arquitectura de Servicios Integrados, fue el primer intento significativo que trató de construir sobre la Internet un modelo de servicios con diferentes calidades. Inicialmente desarrollado y puesto a prueba en el MBONE, abreviación de "multicast backbone" network. MBONE formaba una red "overlay" experimental para probar envío multidifusión sobre redes IP. Esta red estaba construida sobre la Internet contemplando el ruteo de tráfico multicast con QoS para el mismo. Su desarrollo se ubica en los principios de los años 90'.

"The Stones Live in the cyberspace" http://www.savetz.com/mbone/ch7_3.html

En Noviembre de 1994 un concierto de los Rolling Stones en el Cotton Bowl (Dallas, Texas) es considerado el principal concierto multicast en el cyber-espacio ("first major cyberspace multicast concert"). En el concierto los "Rollings" comenzaron cantando "Fade Away" (un cover de Buddy Holly), seguido por "Tumbling Dice" y "You Got Me Rocking". Luego Mick Jagger dio los saludos diciendo algo como: -*"...I wanna say a special welcome to everyone that's, uh, climbed into the Internet tonight and, uh, has got into the MBONE. And I hope it doesn't all collapse..."*, daban la bienvenida y se subían a la Internet, esperando que esto no colapsara.
Trasmitieron por multicast 25 minutos que formaban parte del "Voodoo Lounge tour".

Hasta los 90', la Internet (interconexión de redes IP tal cual fue concebida originalmente) solo ofrecía servicios de mejor-esfuerzo (best-effort). Por otro lado las aplicaciones de tiempo real tienen restricciones de delay y a veces de ancho de banda para las cuales el modelo de envío usando best-effort no es adecuado. En adición a esto la mayoría de estas aplicaciones, RT applications, son montadas sobre el protocolo de transporte UDP, el cual no tiene mecanismos que le permitan reaccionar adecuadamente ante situaciones de congestión. Una gran cantidad de paquetes perdidos harían a la aplicación ser directamente no utilizable, incluso la red podría quedar también casi en un estado fuera de servicio.

Cuando las pruebas multicast fueron llevadas a cabo sobre el MBONE salió a la luz la necesidad de tener mejores políticas de manejo de tráfico. Los experimentos mostraron que un punto técnico importante había sido pasado por alto. La falta de un control explícito de los recursos y la ausencia de administración de tráfico en la red daba como resultado un pobre rendimiento. Los delays variables en las colas de los dispositivos y la pérdida de datagramas indicaban que una arquitectura de QoS era necesaria.

En el modelo ISA un conjunto de mecanismos y protocolos son utilizados para marcar de forma explícita la reserva de los recursos en la red. Para poder obtener un rendimiento esperado desde la red, la aplicación debe solicitar, previamente al envío de datos con necesidades de QoS (a lo largo del camino que posiblemente tomarán

los mensajes), los recursos que considera necesarios.

La reserva de los recursos es lograda describiendo las características del flujo y los requerimientos de recursos de la red. La red puede aceptar la reserva, en el caso que tenga recursos suficientes para acordar la solicitud. Este proceso es llamado **Admisión (Admission)** y se controla mediante el **Control de Admisión (Admission Control)**. Una vez que el flujo es lanzado los nodos de la red deben realizar el control de tráfico mediante la clasificación y el encolado de paquetes.

3.2. Como trabaja ISA, sus pasos

1. Primero la aplicación debe caracterizar el flujo o los flujos de tráfico. A partir de esto indica los requerimientos de QoS y describe el/los flujos. Debe enviar un mensaje para construir el camino que seguirán los datos.

2. El/Los receptor/es envían el requerimiento ISA a la red. El requerimiento será un mensaje de configuración (setup) llevando la especificación del flujo (flow spec) determinado por la aplicación. El flow spec indica la caracterización de los datos y los requerimientos de recursos.

3. Los nodos de la red (e.g. routers), cuando reciben los mensajes de setup de QoS aplicarán el control de admisión. decidiendo si los recursos son suficientes para cubrir el requerimiento. Si los recursos son suficientes, se realiza la reserva y se registra el flujo con un flow id. Si no hay recursos o la red decide no aceptar hará el rechazo.

4. Una vez aceptado el requerimiento y establecido el camino con los recursos reservados, el emisor puede comenzar con la transmisión de los datos.

5. El tráfico atravesará el dominio ISA de QoS y probablemente sufrirá la aplicación de políticas (policing) en los bordes de la red, siendo clasificado, marcado, encolado y/o descartado.

3.3. Componentes del modelo ISA

Las funciones de los nodos de la red pueden ser separados en dos planos de acuerdo con el tipo de información que fluye por la red. Dependiendo de la implementación del router, puede existir una separación de los planos delimitados por elementos de

hardware y software. La división de funcionalidades es realizada con el objetivo de obtener un mejor rendimiento y modularización en favor de la escalabilidad.

Control Plane: funciones relacionadas con la generación de datos de control, necesarias para las decisiones que deben tomar los nodos con el manejo de los datos, payload. Se diagrama el mapa de la red y la información que la define. Por ejemplo, el proceso de ruteo pertenece a este plano. Este proceso alimenta el plano de forwarding, genera la tabla de ruteo, RIB (Routing Information Base), luego a partir de esta se genera una optimización, la FIB (forwarding Information Base). A partir de este plano se decide que paquetes se envían, cuales se descartan y cuales tienen un tratamiento diferenciado. El plano de control es la "inteligencia", "cerebro" de la red.

Data/Forwarding Plane: define la parte de la red responsable de despachar/switchear/ forwardear los datos lo más rápido posible. Toma información del otro plano, analiza los paquetes entrantes y los despacha. El plano de datos usa la FIB (estructura optimizada) en lugar de la RIB. Este tiene una tarea de procesamiento intensivo y, en general, se desarrolla hardware para este propósito, ASIC (Application-specific integrated circuit). El plano de datos se lo puede considerar como el "músculo" de la red.

En términos generales, en la figura 3.1 se muestra la diferencia entre los planos y donde se encontraría la estructura RIB y donde la FIB.

En el documento RFC-1633 es propuesto un framework con un modelo de referencia para una implementación. Este incluye 4 (cuatro) componentes:

- Packet scheduler (planificador)
- Admission control (control de administración)
- Classifier (clasificador)
- Reservation setup protocol (protocolo de reserva)

Se pueden agregar componentes más específicas, como las mencionadas en la tabla que se muestra más adelante. Varios de estos parámetros y componentes se encuentran en otros modelos de QoS, como MPLS o DiffServ. Las marcadas en itálica serán analizadas más adelante.

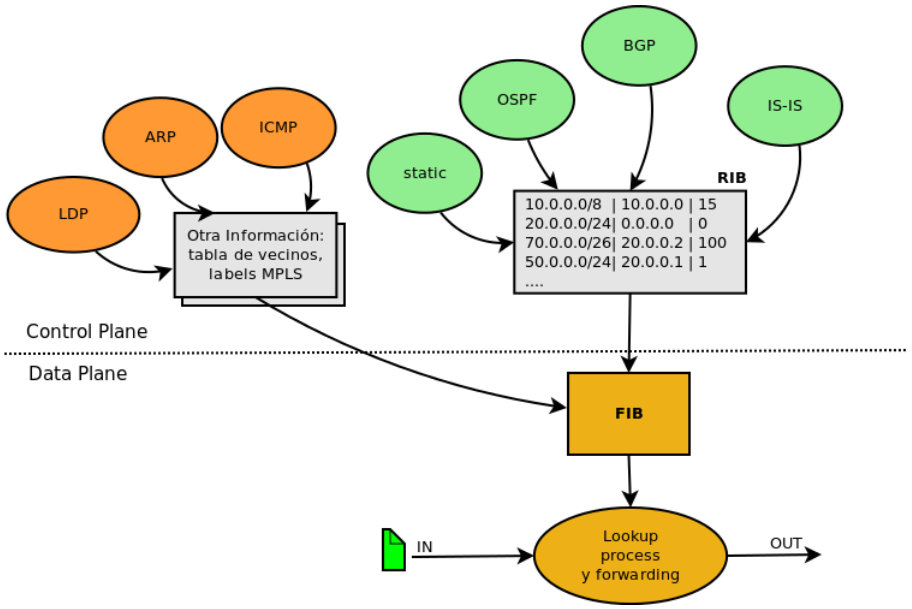


Figura 3.1: RIB vs.FIB

	Data Plane	**Control Plane**
En Hosts	Aplicación con req. de QoS	*Parámetros de QoS requerida* *Resource Reservation Protocol*
En Routers	*Classification*(Flow identificator module) *Encolado* *Planificado de Paquetes /Scheduler (Policing/Shaping)* Switching/Routing Protocol Packets FIB	*Resource Reservation Protocol y Resource Reservation Table* *Admission Control* Protocolo de Ruteo con QoS Protocolos de ruteo tradicionales RIB/Tabla de ruteo

3.3.1. Parámetros de QoS

De acuerdo a RFC-2215 [SW97], los parámetros de QoS pueden ser clasificados en:

Control Parameters: parámetros de control. Usados por las aplicaciones para indicar a la red o dominio de QoS el tratamiento requerido y la caracterización del tráfico.

Characterization Parameters: parámetros de caracterización. Usados para descubrir o indicar las propiedades de QoS del entorno y el estado actual de la

red.

Los siguientes son parámetros para describir los requerimientos de QoS:

Peak rate: tasa pico, el valor mayor al cual el emisor puede enviar datos. A menudo limitado por el hardware, velocidad de la interfaz. Por ejemplo de acuerdo a la tecnología podría ser Fastethernet, 100Mbps; ATM, 622Mbps; Giga, TenGiga, 40G o 100G ethernet: 1Gbps, 10Gbps, 40Gbps, 100Gbps. Para respetar el parámetro el tráfico puede ser shaped (formado) o policed (truncado) -Ver más adelante sección **Policing vs. Shaping**- si es menor que la velocidad de la interfaz.

Average rate: tasa promedio de envío, media de la tasa de tráfico sobre el intervalo de tiempo.

Burst size: la cantidad máxima de bits o bytes que se pueden enviar al **Peak rate**. Describe el grado de transmisión en ráfagas que tiene el tráfico, en inglés **Burstiness**.

Minimum rate: el mínimo ancho de banda digital requerido por la aplicación. El intervalo de tiempo también debe ser especificado.

Delay/Latency: latencia. El tiempo que toma al paquete más pequeño de la aplicación en ir desde el destino hasta el origen. A veces es derivado del RTT (Round Trip Time), tiempo de ida y vuelta. Asociado con requerimientos de tiempo real, RT. El delay tiene dos componentes, una fija: llamada **transmission delay**, que estará formada por:

- Delay de propagación (**propagation delay**) t_{prop}
- Delay de serialización (**serialization delay**) t_{serial}
- Delay de procesamiento (**processing delay**) t_{proc}

El delay de propagación se calcula en base a la velocidad de transmisión con respecto al medio físico y a la distancia. Considerando la velocidad de la luz o de una señal eléctro-magnética y la distancia física que debe recorrer la señal se obtiene la ecuación:

$$t_{prop} = \frac{l}{c \times VOP} \tag{3.1}$$

Para el delay de propagación, donde l es la longitud a recorrer, c la velocidad de la luz y VOP el factor de velocidad de propagación del medio con respecto

Coder	Rate	Required Sample Block	Best Case Coder Delay	Worst Case Coder Delay
ADPCM, G.726	32 Kbps	10 ms	2.5 ms	10 ms
CS-ACELP, G.729A	8.0 Kbps	10 ms	2.5 ms	10 ms
MP-MLQ, G.723.1	6.3 Kbps	30 ms	5 ms	20 ms
MP-ACELP, G.723.1	5.3 Kbps	30 ms	5 ms	20 ms

Figura 3.2: Cuadro de diferentes codecs de audio

a la velocidad de la luz. También nombrado NVP (Nominal Velocity of Propagation). Por ejemplo 1 bit toma 5ms para propagarse por 1000km en una fibra óptica considerando una velocidad de propagación $2 \times 10^8 m/s$, calculada como $c = 3 \times 10^8 m/s$ y un $VOP = 66\,\%$ o 2/3.

La otra componente fija es el delay de serialización, y es calculado como la relación entre el tamaño del paquete en bits, p y la velocidad de la interfaz, r en bit/seg o múltiplos de esta.

$$t_{serial} = \frac{p}{r} \tag{3.2}$$

Por ejemplo transmitir 1500 bytes o 12000 bits en una interfaz Fastethernet de 100Mbps demoraría 0.0012 seg o 1.2 ms.

El delay de procesamiento dependerá de la capacidad del equipo por donde pasará el paquete y se aplicará el switching. Tendrá que ver con el procesador, el manejo de los buffers en memoria y la tecnología de comunicación interna: memoria compartida, bus, crossbar, etc. El hardware es importante, aunque también el software que corre sobre el dispositivo influirán sobre el tiempo del procesamiento del mensaje. En el caso de procesamiento de paquetes de VoIP dependerá por ejemplo del DSP (digital signal processor), del codec (algoritmo de digitalización) y otros factores. Por ejemplo, G.711 requiere menor procesamiento, pero genera data rates de 64Kbps no aptos para algunos enlaces WAN. En la tabla de la figura 3.2 se pueden ver algunos valores indicados por el fabricante cisco de acuerdo a los distintos codecs.

Se puede suponer fija la capacidad y que el tiempo dependerá de la cantidad de bits que deberá procesar. En este caso el factor de procesamiento por bit se expresa en la variable s. Dando finalmente el tiempo fijo como:

$$t_{trans} = t_{prop} + t_{serial} + t_{proc} = \frac{l}{c \times VOP} + \frac{p}{r} + \frac{p}{s} \tag{3.3}$$

La componente variable es el delay de encolado, **queueing delay**, que puede cambiar de acuerdo al estado de la red. Si la red está congestionada demorará más en las colas antes de ser procesado, incluso puede ser descartado. El delay de extremo a extremo o, end-to-end delay, se puede calcular según la ecuación:

$$t_{end2end} = \sum_{i=1}^{n} [(t_{prop} + t_{serial} + t_{proc}) + t_{queue}] \tag{3.4}$$

Donde n son los nodos que debe atravesar y la parte entre paréntesis de la ecuación es la que se puede calcular si se sabe el camino y los dispositivos por los cuales debería pasar, parte fija. Se pueden incluir otras componentes como el **packetization delay** descripto en el libro de Kurose [KR12] donde se indica que, por ejemplo, para VoIP el emisor debe generar los suficientes bits en tiempo para armar el paquete de acuerdo al codec que se utilice, y si no los recibe sufrirá un retraso. También se lo puede relacionar con t_{proc} al asociarlo con la fragmentación, quitar y agregar de encabezados, alineación de campos, de acuerdo al proceso de store-and-forward.

Jitter: diferencia entre el máximo y el mínimo delay. Asociado íntimamente a requerimientos de tiempo real. A menudo este parámetro es más importante que el **delay**. Puede usarse para describir también el tratamiento que puede dar la red.

Loss rate: tasa de pérdida de mensajes. La relación entre todos los paquetes descartados y los transmitidos en un intervalo de tiempo.

Packet Scheduling Algorithm: algoritmo de planificación de paquetes. La forma que el dispositivo intermedio maneja y despacha los paquetes. Cómo este reacciona ante la congestión y la competencia por un mismo recurso.

MTU: Maximum transmission unit, unidad máxima de transferencia, tamaño máximo del paquete en bit o bytes a nivel de protocolo de red, capa 3. En este caso IPv4 o IPv6. Debido a la proliferación de ethernet, sin importar el protocolo de capa de enlace el valor más utilizado es 1500 bytes, aunque suelen encontrarse mayores como el caso de las Jumbotramas o Jumboframes con valores de 9000 bytes.

Formalmente un caracterización de los parámetros es definida en RFC-2215, para el modelo ISA. Los parámetros se asocian con el modelo de servicio con el cual se los identifica. Para la identificación se utiliza el par indicado abajo:

```
<QOS_Param_ID> ::== <Service_model_num, Parameter_num>
```

Para los modelos los parámetros pueden ser:
General Parameters (Parámetros generales) tienen definición común en todos los modelos. Se los asocia con en valor "1" al modelo, el default.

```
<1, parameter_number>
```

De cualquier forma puede ser sobre-escrito por un servicio más específico, valor diferente de "1". Los modelos básicos, aparte del "best-effort", definidos en ISA son los siguientes:

Guaranteed Service Model: Modelo de servicio garantizado. Definido en RFC-2212 [SPG97]. Provee un servicio ligado de manera fija a los parámetros requeridos, por ejemplo, delay de encolado end-to-end. Sirve para implementar garantías de ancho de banda digital y latencia/delay de forma estricta. Por ejemplo se puede usar para requerimientos de QoS para aplicaciones de tiempo real "duro" (hard real-time). Podrían ser usado para emulación de circuitos: CBR (Constant Bit Rate). El número de servicio asignado por la especificación es "2" (dos).

Controlled-load: Service Model: Modelo de servicio de carga controlada. Definido en el RFC-2211 [Wro97b]. Provee un servicio de tratamiento a los flujo de datos con una QoS aproximada a la requerida. Requiere de control de admisión cuando la red está cargada. Es más flexible que el anterior, y, por lo tanto, de menor costo en recursos. El valor asignado por la especificación es "5" (cinco).

Por lo tanto, los modelos son "1", "2" o "5". Los parámetros pueden ser indicados de dos formas:

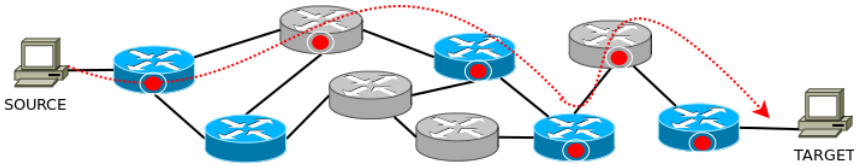


Figura 3.3: Red con el parámetro NUMBER OF IS HOP <1,4,5>

Local: información sobre solo un nodo de la red (i.e. la vista de un router).

Compuesta: un valor que refleja un parámetro a lo largo del camino por donde debe pasar el paquete o flujo. El valor puede ser diferente en cada nodo.

Los parámetros definidos en RFC-2215 son:

`NON-IS_HOP:` IS de "Integrated Services". Valor lógico, si soporta o no QoS con el modelo IntServ. Por ejemplo, un nodo "best-effort" no cumpliría la condición.

```
<_,1,{0|1}>
```

En forma compuesta podría indicar que algunos nodos no lo soportan. El código para el parámetro individual es "1'(uno) y compuesto "2". Por ejemplo:

```
<2,2>
```

indica que algún nodo no soporta el modelo garantizado.

`NUMBER_OF_IS_HOPS:` También conocido como `IS_HOP_COUNT`. Valor compuesto que indica los nodos que soportan el modelo IntServ especificado. El código para el parámetro es "4". Por ejemplo el siguiente valor sería una representación indicando que hay 5 (cinco) nodos que soportan el entorno, framework ISA, sin importar que modelo. Donde podría haber más de 5. Se cuenta el origen. Ver figura 3.3.

```
<1,4,5>
```

El valor máximo es 255 por el hop-count. El código individual es "3", aunque no tiene sentido, ya que es un valor compuesto. La función de agregación es COUNT-IF-ISA.

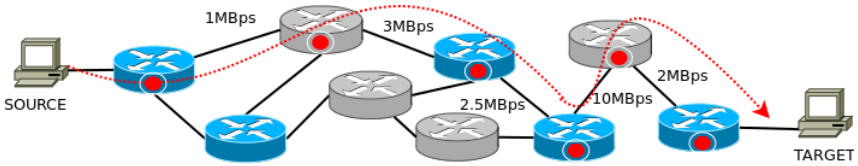


Figura 3.4: Red con el parámetro AVAIL_PATH_BW <1,6,1000000>

`AVAILABLE_PATH_BANDWIDTH:` refleja el ancho de banda digital disponible. El máximo 40Tbps (máximo teórico que se puede obtener en un pelo de fibra óptica). El código asignado es "5" de forma local y "6" de forma compuesta. Está dado en bytes por segundo. De forma compuesta la función de agregación es el MIN. Por ejemplo para la figura 3.4 indica que el valor mínimo a lo largo del camino es de 1MBps.

```
<1,6,1000000>
```

Un servicio técnicamente con restricciones impuestas por configuración del administrador, por ejemplo policing, podría describir el valor que ofrece y no el real.

`MINIMUM_PATH_LATENCY:` debe reflejar la mínima latencia que sufrirían los paquetes de un mismo flujo a lo largo del camino. Valor compuesto por varias componentes como se indicó anteriormente. Solo las componentes fijas se indican, el Queueing delay no debe ser considerado. Debería subestimar la latencia. No se provee un máximo. El código local es "7" y el global "8". En este caso la función de agregación es SUM. La unidad usada es el μs (microsegundo). El valor máximo es $2^{32}-2$. El valor $2^{32}-1$ tiene como significado: latencia indeterminada.

`PATH_MTU:` computa el MTU mínimo en el camino por donde debe pasar el tráfico. El PMTU no se puede usar, pues solo toma en cuenta el modelo "1". Código individual "9" y compuesto "10". Medido en bytes, donde el mayor es valor es $2^{32}-1$ bytes, lo cual es suficiente como para tener en cuenta los "Jumbogramas" IPv6, definidos en RFC-2675.

`TOKEN_BUCKET_TSPEC:` sirve para indicar por el emisor como será la generación de tráfico. Tiene asignado el código "127". No es un parámetro de caracterización. Es una estructura de datos usada por el emisor y los nodos de borde del dominio

de QoS. Basado en la "token bucket metaphor".

Cómo Trabaja un Token Bucket: Se define como la tula (r, b)

1. Los token son puesto en el bucket a la tasa, velocidad: r.
2. La capacidad máxima del bucket de tokens es: b.
3. Cada token representa el permiso para enviar una cantidad de bytes, donde en general $1tk = 1byte$.
4. Cuando se envían N bytes se remueven la cantidad de tokens que representan cada byte.
5. Si no hay suficientes tokens el paquete debe esperar o es descartado, de acuerdo a la política.
6. Sobre un período de tiempo T la cantidad de datos enviados no excederá: $(r \times T) + b$ si suponemos $1tk = 1byte$.

De acuerdo al RFC-2115 los parámetro descriptos son:

Token rate (r): (Bps), max 40TBps. relacionado con `AVAILABLE_PATH_BANDWIDTH`.

Bucket depth (b): (B), max 250GB. tamaño máximo del bucket.

Peak rate (p): (Bps), max 40TBps. tasa máxima de transmisión.

Minimum policed unit (m): (B) todos los paquetes/datagramas menores o iguales al tamaño m son tratados como de tamaño m.

Maximum packet size (M): (B) el tamaño máximo del paquete. Los que son de tamaño mayor no se consideran como parte del flujo que requiere la QoS.

Un modelo genérico solo usa 3 de los 5 parámetros. Tamaño de ráfaga (B_c), tasa de envío promedio(M_r) e intervalo de tiempo (T_c). A veces se miden en bits y otras en bytes.

$$M_r = \frac{B_c}{T_c} \tag{3.5}$$

M_r, llamado también committed information rate (CIR). El valor T_c se puede definir de acuerdo al equipo. Algunos fabricantes lo permiten en el rango de 10ms a 125ms. Este método de acuerdo a la capacidad del token permite el envío de ráfagas en exceso, dando la velocidad máxima para el flujo como la ecuación 3.6.

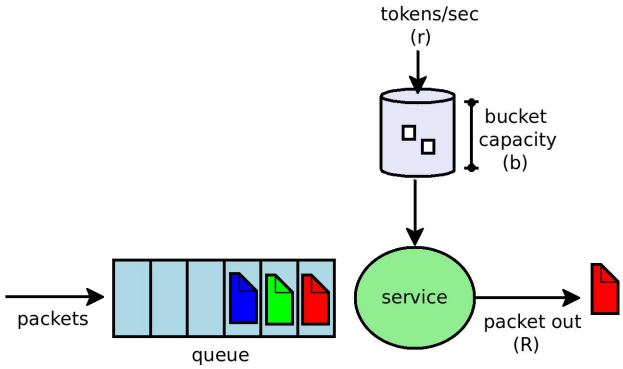


Figura 3.5: Ilustración de la metáfora del token bucket

$$\frac{b \times 8}{T_c} + C_r \tag{3.6}$$

Donde b es la capacidad del bucket, C_r es la velocidad establecida en bps. A largo plazo este método controla que no se exceda de la velocidad r. El funcionamiento del token bucket se ilustra en la figura 3.5.

Este mismo mecanismo se puede utilizar para tratar de dos forma diferente al tráfico en exceso:

- Descartarlo (aplicar un policer)
- Demorarlo almacenándolo (aplicar un shaper)

3.3.2. Policing vs Shaping

Cuando el tráfico es **"policed"** (truncado), los paquetes que arriban fuera del límite establecido son descartados o potencialmente marcados. En contraste, cuando es **"shapeado"** (formado), el tráfico en exceso es demorado en colas de salida. Gráficamente el policing se puede ver como dientes de sierra con mesetas y caídas (saw-tooth with plateaus and valleys). El shaping en cambio, en casos que el tráfico siempre esté en exceso, se verá como una línea constante sobre el límite.

Shaping implica la existencia de una cola en memoria para almacenar los paquetes demorados, mientras que un **Policer** no. Un policer puede ser aplicado al tráfico de entrada como al de salida. el shaper solo al de salida. De acuerdo a los parámetros de delay, optimización del ancho de banda y tráfico en ráfagas se aplicará uno u otro. En la figura 3.6, tomada de la documentación de cisco se ilustran los dos mecanismos.

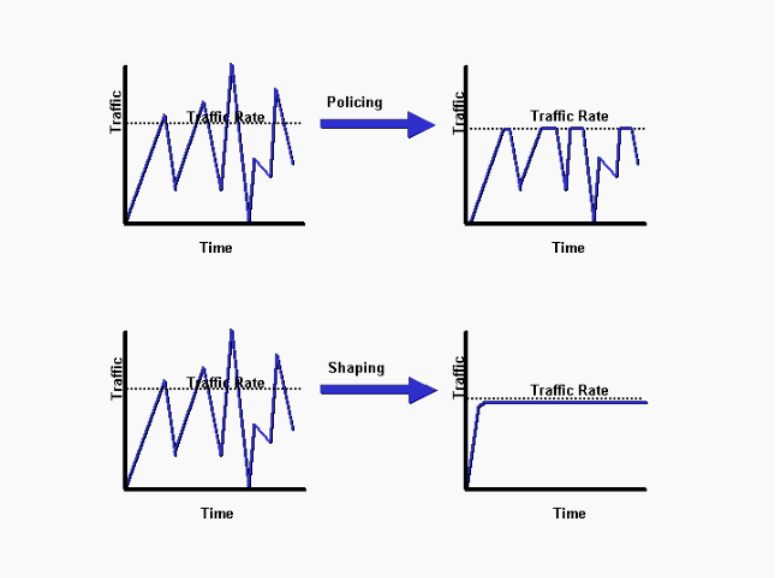


Figura 3.6: Diagrama que muestra diferencia entre shaping y policing

3.3.3. RSVP (Resource Reservation Protocol)

Previamente a utilizar los recursos, un proceso de reserva es necesario. RSVP (Resource Reservation Protocol) es el nombre del protocolo implementado para tal propósito en el modelo IntServ. Este fue diseñado como un protocolo de señalización y es un protocolo de control separado del resto de los servicios ISA, lo cual le permite ser utilizado por otros modelo y protocolos, por ejemplo en MPLS con MPLS-TE. RSVP es un protocolo de control para IP como lo son ICMP, IGMP, ICMPv6, RIP, OSPF, BGP, LDP, SIP y tantos otros.

3.3.3.1. Operación de RSVP

1. El emisor potencial es el que inicia el proceso de reserva. Solicita las recursos mediante paquetes de reserva hacia el receptor o receptores. Es un **Path message** que sigue el flujo downstream hacia el/los receptores.

2. Cada nodo intermedio (i.e: routers en el camino del mensaje) procesara el requerimiento agregando el mensaje a sus capacidades de QoS.

3. El mensaje puede estar destinado a un solo receptor o un grupo, por ejemplo el caso de una dirección de multicast destino. Previo a esto se debe solicitar la unión al grupo mediante IGMP en IPv4 , o ICMP Multicast Listener Discovery (MLD) en IPv6.

4. El receptor recibirá el **Path message**. Una vez recibido procesa la QoS solicitada y genera un paquete que contiene un **Reservation message** en sentido contrarío, upstream hasta el emisor. Es importante notar que la reserva la hace el receptor y no el emisor, aunque el proceso es disparado por el emisor.

5. El mensaje de reserva deberá seguir el camino marcado por el **Path message**. En cada nodo se harán las reservas.

6. Cuando el emisor recibe el mensaje, se entiende que se reservaron los recursos, por lo que puede comenzar a enviar datos por la aplicación específica.

7. Las reservas de los recursos deben ser refrescadas por los receptores con mensajes para este propósito. Se debe refrescar las reservas distribuidas en toda la red.

8. Si el receptor no tiene más interés en mantener las reservas, puede darlas de baja enviando al nodo remoto un **Teardown message**.

La reserva de los recursos tiene las siguientes características:

- Simplex reservation, RSVP reserva los recursos solo en una dirección (simplex flow).
- Receiver oriented, orientada por el receptor, los receptores deciden que recursos reservan e inician el proceso previo a los datos de la aplicación del emisor. Si bien el mensaje inicial lo genera el emisor.
- Los mensajes RSVP son transportados directamente en IP y no implementan un sub-protocol de control, por lo cual no tienen un despacho confiable.
- RSVP es independiente de las políticas de ruteo.
- RSVP trabaja para unicast y multicast.
- RSVP genera estados en los nodos intermedios donde se reservan los recursos. Estos se llaman **Soft States**.

3.3.4. Mensajes RSVP

3.3.4.1. Identificador de sesión

Una sesión RSVP es identificada por un ID global formado por la dirección destino más 32 bit llamados "reservation ID" definido en RFC-2205 donde se indica: `(ProtocolId [, DstPort])`. Por lo que el "Session Identification" total será:

```
(DestAddress, ProtocolId [, DstPort])
```

El mecanismo para obtener o salvar estos valores de un repositorio no está definido, podría ser, por ejemplo, un servicio de directorio similar a LDAP o una DB relacional.

3.3.4.2. RSVP Path Message

Un **Path message** es enviado por el potencial emisor de la sesión para marcar el camino en la red. El destino pude ser uno o varios equipos (caso de multicast). Las direcciones IP del mensaje deben ser las mismas que llevará el tráfico de datos. Este mensaje de control contiene:

- Sender Template.
- Sender Tspec.
- Adspec.

El **Sender Template** especifica la dirección del emisor, y opcionalmente, el port origen. Es como un **Filterspec** reducido (ver detalles más adelante).

El **Sender Tspec (Traffic Specification)** indica las características del tráfico, flujo de datos a enviar. No debe poder ser modificada por nodos intermedios. Especificada en RFC-2210 [Wro97a]. Podría contener por ejemplo el parámetro global `TOKEN_BUCKET_TPSEC <1,127>`.

El parámetro **Adspec (Advertising Information specification)**, definido en RFC-2210, lleva la información que anuncia como debería ser tratado el tráfico en los nodos por los que pasará. Es información para el receptor, a partir de la cual puede predecir el servicio que se obtendrá end-to-end, por ejemplo si el modelo es garantizado (Guaranteed) o de carga controlada (Controlled Load) En cada nodo intermedio por el cual pasa el mensaje, los parámetros son actualizados. En el mismo se cargan la cantidad de nodos que soportan IntServ en el parámetro `NUMBER_OF_IS_HOPS`, el

MTU compuesto, el ancho de banda digital compuesto y otros.

Los mensajes de generación de camino, **Path messages**, producen información para el RPF (Reverse Path Forwarding) en cada nodo intermedio. Básicamente las direcciones IP unicast hop-by-hop, que luego serán usadas por los mensajes de reserva **Reservation messages**. Debido a la carencia de un protocolo de ruteo específico para RSVP se requiere mantener las IP del próximo salto en el camino en cada nodo.

3.3.4.3. RSVP Reservation Message

Los mensajes de reserva **Reservation messages** o **Resv messages**, son enviados por los receptores de la sesión. El destino es el emisor. Son enviados hop-by-hop por cada nodo que soporta RSVP en el camino usando la información de RPF que se generó con los mensajes de camino, **Path messages**. Los mensajes de reserva son unicast a cada dirección del próximo salto de acuerdo al valor de RPF almacenado.

Un requerimiento de reserva llevado en estos mensajes consiste de un par: **Flowspec** y un **Filterspec**, llamada esta tupla: **Flow Descriptor**. Los recursos solicitados se indican en el Flowspec y el tráfico afectado en el Filterspec. También se debe especificar el estilo de reserva.

Filterspec: de acuerdo a RFC-2205 es usado para distinguir los paquetes que conformarán el flujo de la sesión, es decir los paquetes a recibir la QoS requerida en el parámetro Flowspec. Dependerá del protocolo usado, por ejemplo las direcciones origen y destino, los puertos en el caso de usar TCP o UDP, el DSCP o Traffic Class, etc. Para el caso de IPv6 se puede usar el campo Flow Label para una eficiente clasificación. Estos valores deben ser analizados por cada nodo intermedio para aplicar el tratamiento, situación que podría no ser posible, por ejemplo, en presencia de IPSec.

Flowspec: Definida en RFC-2110, en este parámetro se indican los recursos a reservar para todos o algunos nodos de la red, que serán aplicados luego a los paquetes que concuerden con el Filterspec. Puede estar formado por parámetros como delay, ancho de banda digital, jitter o `TOKEN_BUCKET_SPEC`. El Flowspec, en el caso de ser un servicio garantizado, tendrá un **Rspec** ("R" de reserve) y un **Tspec**. En el caso de ser un controlled-load service solo el **Tspec** será indicado.

Estilos de reserva de RSVP (reservation "styles"): RSVP define diferentes formas de indicar los Filterspec, por ejemplo con wildcards, con filtros fijos o con filtros dinámicos.

Si el proceso de admisión es aceptado, el Flowspec será usado para parametrizar una clase de recursos en el **Planificador (Packet Scheduler)** y el Filterspec será instanciado en el **Clasificador de paquetes (packet classifier)**.

3.3.4.4. RSVP Error Message

Existen dos mensajes de errores en RSVP, **ResvErr** y **PathErr**. Estos últimos son enviados al emisor que genera el error. No cambian el estado en los nodos por los que pasa. Indica un error en el procesamiento de los mensajes para establecer le camino.

ResvErr son enviados como error al procesar el mensaje de reserva. Son enviados de vuelta a los receptores pasando por cada nodo del camino conformado. Algunos ejemplos son:

- `Code 0x01`, falló el control de admisión, por ejemplo por falta de recursos. Los Sub-code de los errores pueden ser.

  1: El límite del Delay requerido no se puede cumplir.

  2: El ancho de banda digital requerido no está disponible.

  3: El MTU del requerimiento es mayor que el del camino.

- `Code 0x15`, error en el formato del requerimiento, por ejemplo se mezclan parámetros de diferentes servicios incompatibles o el servicio solicitado no es soportado. Otros podrían ser que el Flowspec, Tspec o Adspec no tienen el formato esperado.

3.3.4.5. RSVP Confirmation Message

Cuando se realiza la reserva, esta debe ser confirmada con un mensaje **ResvConf message** que incluye el **Resv message**. Estos mensajes son enviados salto a salto hasta el receptor que inició la reserva. Usados para confirman la reserva exitosa. En caso de no poder satisfacerla se generará un **ResvErr**, por ejemplo cuando el Flowspec es mayor que los soportados por los nodos de la red.

3.3.4.6. RSVP Teardown Message

Un **Teardown message** es básicamente una indicación de remover los estados y las reserva de recursos en los nodos que brindan el servicio a lo largo del camino establecido. Si los estados no son refrescados por **Resv messages** serán removidos sin necesidad de estos mensajes. Son usados para no esperar los timeouts y permitir un mejor aprovechamiento de los recursos. Pueden ser generados por el emisor o por el/los receptores. Dentro de esta clase de mensajes hay dos: **PathTear** y **ResvTear**. Los primeros van por todo el camino de extremo a extremo desde el emisor indicando nodo a nodo la liberación de los recursos. A diferencia, los **ResvTear** van salto a salto, downstream desde el/los nodos de recepción.

3.3.4.7. Soft State (Estados Dúctiles)

De acuerdo al documento RFC-2205 existen dos posibles estilos de mantener la información de reserva de recursos en los nodos:

- 'hard state" (HS), "connection oriented"
- "soft state" (SS), "connectionless"

Los estados HS son manejados de manera completamente determinista en cooperación con la red. La red tiene la responsabilidad de crearlos. Funcionarían como el manejo de circuitos virtuales permanentes, PVC.

La alternativa usada por RSVP son los "soft states" (SS), método por el cual los recursos son creados, cacheados, refrescados y dados de baja de forma dinámica. Si el timer de refresco, TT expira, son dados de baja automáticamente. La tasa de refresco está indicado por el valor RR, siendo $RR << TT$ (mucho menor). Este método de mantener los estados es más robusto y flexible a los cambios de la red, en particular sobre una red IP donde el tráfico es "connectionless". Haciendo una analogía con los circuitos virtuales sería como establecer SVC (Switched Virtual Circuits), circuitos bajo demanda.

3.3.5. Control de Admisión

El **Control de Admisión** debe ser implementado en cada nodo de la arquitectura. A través de este se provee la decisión de soportar o no un requerimiento de QoS para un nuevo flujo de datos. La decisión se toma de forma local, ADMIT o

REJECT, para el nuevo flujo. En caso de rechazo, siempre debe existir la posibilidad de transmitirlo con el servicio default, "best-effort".

Para poder garantizar la QoS, se debe supervisar el uso de los recursos por los flujos activos. El monitoreo es parte importante de la tarea del módulo de control de admisión.

Dos formas de implementar el control de admisión son mencionadas en la referencia [ZC01]:

Parameter-based: basada en parámetros. Un conjunto de parámetros son usados para especificar de forma precisa las características del tráfico. Se hace el cálculo en base a los parámetros indicados. En general no es fácil caracterizar antes de generar realmente los datos.

Measurement-based: basado en mediciones. En lugar de confiar en una caracterización previa, la red se encarga de medir e indicar las características reales. Se usan enfoques estadísticos para poder generalizar.

3.3.6. Clasificación de Paquetes

La clasificación de paquetes y la asociación con los flujos de datos se hace mediante un mapping con la clase de servicio requerida. Los paquetes pertenecientes al mismo flujo deben ser identificados y clasificados de acuerdo a la QoS solicitada. En términos de la arquitectura IntServ la detección del flujo y su identificación debe realizarse en cada nodo por cada paquete. Para esta tarea se requiere inspeccionar los encabezados de los mismos. Esta labor se la llama habitualmente detección del Flow ID.

Debido a que en IPv4 no se tiene un campo para este propósito los campos de los encabezados de control deben inspeccionarse y utilizarse para la clasificación. El caso más común es usar la 5-tupla: Dirección Origen, Dirección Destino, Protocolo, Puerto Origen, Puerto Destino. Para el caso de los protocolos que no usen puertos para la multiplexación, como puede ser ICMP o GRE solo se usa la 3-tupla: Dirección Origen, Dirección Destino, Protocolo; pudiéndose sumar otros campos, por ejemplo para GRE, RFC-2890, se tiene el campo Key, que permite identificar tráfico de diferentes flujos transportado dentro del mismo túnel.

En IPv6 se encuentra el campo para este propósito, **Flow Label**, que, junto con las direcciones serviría para conformar la 3-tupla de identificación: Dirección Origen, Dirección Destino, Flow Label. Esto evita que la inspección tenga que inmiscuirse en los encabezados de transporte, dando probablemente una mejor performance. El tratamiento de este campo se verá en capítulos específicos más adelante en este texto.

Una vez que el paquete fue identificado con un flujo, puede ser clasificado de acuerdo al mapping flow ID $\rightarrow$ Clase. Esta tarea se delega a un módulo llamado **Clasificador (Classifier)**. Todos los paquetes de la misma clase recibirán el mismo tratamiento. Las relaciones entre clases y tratamientos son abstracciones que pueden ser locales, dando como resultado posible diferentes tratamientos de acuerdo al nodo por donde pasa el paquete. Por ejemplo, puede suceder que en los routers de núcleo o backbone de una red, varios flujos son asociados con una misma clase, haciendo agregación, en cambio los routers de borde o edge podrían dividirlos en diferentes clases.

3.3.7. Encolado y planificación de paquetes

Una vez que la identificación y clasificación fue realizada, el tratamiento dependerá de las tabla de reserva mantenida por los Soft States. Probablemente la etapa más importante en una arquitectura de QoS es la planificación de los paquetes (packet scheduling). Esta será la responsable de hacer un buen manejo de los recursos para cumplir con los requerimientos indicados. El planificador trabaja administrando el despacho de los paquetes de los flujos, asistido por estructuras de datos como son las colas.

Una forma sencilla de ver un planificador es asociarlo con una implementación de un token bucket. Los paquetes son recibidos en la cola de entrada INPUT-Q, son luego clasificados, a continuación se selecciona la interfaz de salida, por ejemplo mediante la FIB, para pasar a la/las colas asociadas con esta. Las OUTPUT-Q pueden estar implementadas con diferentes estrategias, como FIFO(FCFS), CBQ-WFQ, LLQ, PRIORITY-Q u otras. Las estrategias dependerán de la planificación. Los paquetes pueden ser re-ordenados, re-marcados, demorados, descartados y/o enviados. El planificador puede aplicar un policer o un shaper sobre el tráfico.

Para el método de descarte se pueden definir diferentes políticas, por ejemplo puede ser una simple como descartar los que llegan cuando no hay más espacio en el

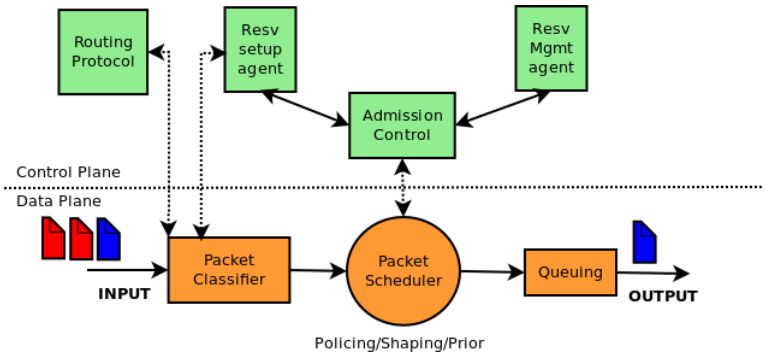


Figura 3.7: Diagrama de componentes del modelo ISA

buffer o este se está por agotar: **Tail Drop**; o puede tener un algoritmo de selección de que paquetes descartar de acuerdo a alguna marca. Otra técnica es la de descartar de forma aleatoria luego de que se alcanza un porcentaje de paquetes encolados: **RED**, o de forma más inteligente mirando la prioridad de los mensajes: **WRED**. Las políticas de encolado y descarte son estudiadas en el capítulo "Estrategias de Encolado".

Dentro del RFC-1633 otra componente a tener en cuenta, como parte del planificador, es el **Estimador (Estimator)**. Este es el algoritmo usado para medir las propiedades del tráfico de salida y desarrollar estadísticas para controlar el planificador y el control de admisión. En la figura 3.7 se muestra la relación entre todas estas componentes.

3.4. Modelos de Servicio ISA

Además del modelo "best-effort" (default) los servicios ISA describen otros dos modelos donde se contempla la QoS. Se vieron ya las características cuando se estudió la parametrización los esquemas: **Guaranteed Service** y **Controlled-load Service**.

3.4.1. Modelo Guaranteed Service

En RFC-2212 se describe este modelo de servicio el cual provee firmes restricciones de forma end-to-end. Para que se pueda alcanzar todos los nodos en el camino deben soportarlo. Los paquetes del flujo que obtienen este servicio no deben ser descartados debido al overflow de las colas y deben permanecer dentro de los parámetros solicitados.

El servicio no controla la latencia mínima promedio pero sí el tiempo de encolado máximo, es decir el delay en las colas. Los paquetes podrían obtener un menor delay, no considera el jitter. Las aplicaciones pueden considerar esto haciendo "buffering" del lado del receptor.

Los enlaces por los que pasa el tráfico no deben fragmentar y, si supera el MTU por los paquetes serán tratados como tráfico fuera del flujo (non-conformant traffic). En los bordes de la red, mediante policers, el tráfico debe ser forzado a cumplir con los parámetros que lo caracterizan. Si no se cumple puede ser marcado y transmitido para ser tratado como "best-effort".

El requerimiento del servicio se hace con un Flowspec de dos parámetros: un `Tspec` que caracteriza el tráfico mediante un `TOKEN_BUCKET_TSPEC` y un `Rspec` para indicar el servicio a la red. Este último se conforma de:

Service rate (R): (Bps). el ancho de banda digital necesario. R debe ser mayor o igual a r (token bucket rate).

Slack Term (S): (ms). el tiempo extra de delay que los nodos pueden agregar al tiempo total end-to-end. El valor no puede ser negativo.

El Rspec rate puede ser mayor que el TSpec como se indicó, así valores mayores pueden reducir la necesidad de encolado. El parámetro Rspec tiene asignado el código "130".

El servicio se puede ver como un VC (circuito virtual) con un ancho de banda digital garantizado. No debería producirse encolado con descarte. Se asume que no hay re-routing ni tampoco falla de enlaces.

Los peores casos para definir Tspec y Rspec de forma end-to-end pueden ser calculados de forma matemática. Si se tiene un Token bucket con los parámetros (r, b, p) donde el caso ideal es velocidad de salida de interfaz infinita, p.

$$QDelay = \frac{b}{R}, p \to \infty, R \geq r \tag{3.7}$$

Y cuando p no es infinito:

$$QDelay = \frac{b(p-R)}{R(p-r)}, p > R \geq r \tag{3.8}$$

Si no se considera el caso perfecto, se agregan dos errores. El término C, el delay que sufrirán los mensajes debido a las velocidades de transmisión y los diferentes tamaños de mensajes. El otro es D, que es independiente de las velocidades, y representa el peor caso sin tener en cuenta la velocidad. Este delay se puede asociar al pipelining de procesamiento en el router. Cada nodo introducirá una cantidad fija al valor D_{tot}. El valor D se calcula para cada salto. Las sumas de C y D en todo el camino son C_{tot} y D_{tot}. También se consideran como parámetros las sumas parciales D_{sum} y C_{sum}. Los parámetros para el modelo ISA tienen asignados los códigos:

```
C    <2, 131>
D    <2, 132>

Ctot <2, 133>
Dtot <2, 134>

Csum <2, 135>
Dsum <2, 136>
```

Para el cálculo del buffer , se define el término M, el tamaño máximo de paquete o datagrama:

$$QDelay = \frac{(b-M)}{R(p-R)}(p-r) + \frac{M+C_{tot}}{R+D_{tot}}, p > R \geq r \tag{3.9}$$

$$QDelay = \frac{(M+C_{tot})}{R+D_{tot}}, R \geq p \geq r \tag{3.10}$$

El tamaño del buffer para los nodos intermedios se calcula entonces como:

$$QLength = b + C_{sum} + D_{sum} \times r \tag{3.11}$$

Lo que se controla es el delay de encolado, el resto deberían ser fijos, bajo las condiciones estables.

3.4.2. Modelo Controlled-load Service

El otro modelo definido para ISA es definido en RFC-2210. Tiene las características de la QoS que obtendría un flujo sobre una red sin carga, "best-effort" en una red sin congestión. El modelo, como se indicó recibe un tratamiento menos estricto que el anterior, por ejemplo se asimila a un STDM (statistical multiplexing) con mejores

garantías a menor costo de recursos que el servicio garantizado.

En este tampoco se permite fragmentar y, sucede de forma similar que con el anterior, el tratamiento de los mensajes que no cumplen con el MTU serán tratados como no pertenecientes al flujo.

La solicitud de este servicio solo requiere un parámetro de Flowspec mediante un Tspec con un `TOKEN_BUCKET_TSPEC`. La aplicación puede asumir una muy baja tasa de descartes y que el delay no sobrepasará por mucho el mínimo de la red.

Es un modelo apto como se mencionó para STM, como tráfico elástico o por ráfagas que puede tolerar pérdidas y delay dentro de límites razonables, por ejemplo aplicaciones de real-time soft, adaptativas. Un caso particular podría ser SNA sobre TCP/IP.

3.5. IntServ en IPv6

De acuerdo a "IPv6 in Practice" [Sto06] el uso del campo de Flow Label ayudaría al desarrollo de IntServ para IPv6, aunque no parece haber tenido desarrollos aplicables en entornos reales. El problema de la escalabilidad de IntServ debido a la gran cantidad de flujos y estados generados se intenta atacar en RFC-3175 "Aggregation of RSVP for IPv4 and IPv6 Reservations" [BILFD01], tratando de agregar flujos reduciendo así el número de estados y recursos que se producen individualmente, aunque la cantidad de paquetes pasando por los routers de tránsito sigue siendo importante, limitando la escalabilidad. El autor del documento citado indica que no tiene presencia de ninguna implementación de IntServ para IPv6.

En el libro "IPv6 Essential" [Hag06] indica que IntServ combinado con RSVP puede ser complejo de implementar y remarca la limitación de la escalabilidad, haciéndolo inadecuado para ofrecer QoS en una red global como Internet.

En "IPv6, 2nd Edition" [Los03], se indica que IntServ ha probado ser inadecuado como una solución individual para ofrecer QoS, no escala en un entorno global y es complicado de implementar.

Si bien varias RFC del IETF trabajan sobre IntServ sobre IPv4 como sobre IPv6, por ejemplo RFC-2205, RFC-2207, RFC-3175 su desarrollo en aspectos prácticos estrictamente sobre IPv6 parece estar detenido y los avances se concentran en la señalización para MPLS o GMPLS (Generalized MultiProtocol Label Switching) usando RSVP-TE (RSVP Traffic Engineering) definido en RFC-3209 y luego actualizada por otros documentos. Una buena fuente de referencias a desarrollos y parámetros de IETF de RSVP se puede obtener en:

- `http://www.networksorcery.com/enp/protocol/rsvp.htm`
- `http://www.iana.org/assignments/integ-serv/integ-serv.xhtml`

Hoy los fabricantes tienen soluciones IntServ con RSVP para IPv4, pero no para montarlo directo en IPv6. Por ejemplo en la documentación de cisco en su solución para VoIP indica en inglés: *"-...RSVP does not support IPv6. RSVP calls support IPv4. If RSVP is required for the call and any device in the call is configured for or uses an IPv6 address, Cisco Unified Communications Manager rejects the call, and the caller receives a busy tone..."*.

3.6. Ejemplo con IntServ

Finalizado el estudio de IntServ y RSVP tratados en este capítulo se puede consultar en el segundo apéndice, "B", el ejemplo que muestra IntServ con RSVP en acción.

3.7. Conclusiones sobre IntServ

IntServ es una arquitectura de QoS que ofrece un marco para que las aplicaciones puedan obtener de la red el servicio deseable, sin embargo la arquitectura tiene muchos inconvenientes. El problema más importante es la falta de escalabilidad, como se mencionan en [FPRS03].

El problema de escalabilidad se ve en varios puntos:

- En primer lugar, la calidad de servicio no puede ser soportada a menos que todos los nodos de la red implementan IntServ. Cada router debe implementar señalización RSVP, control de admisión, clasificación y planificación de tráfico. Puede ser implementada parcialmente, no en todos los nodos, pero no todos los modelos pueden ser ofrecidos en este caso.

- IntServ clasifica paquetes en un gran número de clases, lo hace por flujo, esto aumenta el número de "clases" generando problemas de rendimiento en la mayoría de los nodos de la red, principalmente en el núcleo.
- La implementación de IntServ en el núcleo de Internet es impracticable debido al enorme almacenamiento necesario para los estados y la alta carga de procesamiento que se generaría. Para que sea útil IntServ de forma end-to-end requiere acuerdos entre diferentes redes, diferentes vendedores de equipos y normas más estrictas. Si bien RSVP se puede implementar dentro de una red privada, empresa/organización mediana, es difícil llevarlo a grandes redes públicas.
- Por último, cada solicitud de reserva necesita intercambiar una cantidad no trivial de mensajes para la señalización, recursos de procesamiento y memoria en cada nodo. Todos los estados se deben refrescar de forma periódica, afectando de forma importante la escalabilidad.

Un punto que no queda cubierto por la arquitectura es como autorizar las solicitudes y como priorizar entre diferentes requerimientos al momento de aceptar o rechazar. Hoy en día no hay muchas aplicaciones que soporten RSVP por lo que no ha sido ampliamente difundido su uso. Los routers que soportan IntServ deben tener una cantidad significativa de ancho de banda para ofrecer, pues esta arquitectura no fue diseñada para trabajar en enlaces de baja velocidad.

IntServ se puede encontrar hoy en día en las redes corporativas que trabajan para un ancho de banda y delay garantizado, por ejemplo, para telefonía sobre IP o vídeo conferencia. Aunque su uso ha disminuido, muchas de las ideas y conceptos desarrollados se encuentran en las nuevas arquitecturas de QoS, por ejemplo la idea del servicio de carga controlada ayudó a desarrollar la arquitectura DiffServ, o el protocolo RSVP se utiliza hoy con MPLS o GMPLS Traffic Engineering con RSVP-TE. Con respecto a RSVP y RSVP-TE en el documento IETF RFC-4094 "Analysis of Existing Quality-of-Service Signaling Protocols" [MF05] se hace un análisis de estos y otros protocolos de señalización para QoS.

Capítulo 4

Estrategias de encolado

4.1. Introducción

Las estrategias de encolado y de descarte de paquetes son fundamentales para el funcionamiento de la QoS. Sin estas, es imposible ofrecer un tratamiento diferenciado a los distintos tipos de tráfico. Las estrategias de encolado se utilizan en ambos modelos, tanto DiffServ como IntServ. La elección de la disciplina correcta para armar las colas y la longitud de las mismas son tareas cruciales en el proceso de definición del tratamiento con QoS. Cuando se selecciona la longitud de la cola, esta debe ser la adecuada, pues, si de alguna manera es demasiado corta podría desechar fácilmente paquetes, o, si es demasiado larga, se podría añadir una cantidad de latencia o jitter que las aplicaciones no podrían tolerar. De acuerdo a las colas, las sesiones de los programas montados sobre determinados protocolos, por ejemplo: TCP, podrían no funcionar de extremo a extremo si los paquetes tienen que esperar largos períodos de tiempo en las colas. Dentro de las tareas para ofrecer QoS, es crucial entender las capacidades de encolamiento de los routers en los nodos intermedios.

En este capítulo se describirán varios de los mecanismos disponibles en los nodos de red para el encolado y descarte de paquetes. En el apéndice asociado se darán algunos ejemplos prácticos proporcionando una visión general de varios de los métodos que se encuentran en la mayoría de los dispositivos de red actuales.

4.2. Breve introducción a la teoría de colas

Cualquier sistema de servicio compartido puede ser caracterizado por un proceso de llegada y otro de salida (proceso de servicio). En las redes de datos, los paquetes o

los datagramas son los elementos a ser servidos, son los que llegan y salen de los nodos.

Un router es un recurso compartido en la red que ofrece el servicio de enrutamiento de tráfico. Los paquetes arriban al sistema (router) con determinada distribución de probabilidad, en el proceso de enrutamiento son servidos; el router elige el siguiente salto y realiza las tareas adicionales como decrementar el hop-limit. Finalmente serán despachados por una interfaz de salida. Cuando un paquete llega al router y este está ocupado debido, por ejemplo, al procesamiento de otro paquete, el entrante deberá esperar, por lo que la cola ahora tendrá un nuevo elemento.

El comportamiento del sistema de encolado es dependiente de:

- Factores externos:
  - Proceso de llegada, arribo (la tasa y la distribución de los tiempos entre llegadas).
  - Proceso de servicio, despacho (la duración media y la distribución de los tiempos de servicio). Esta dependerá de los tamaños y requerimientos de cada paquete a ser atendido.
  - El tamaño de la población a ser servida.
- Características del mismo sistema de colas:
  - El número de servidores, proceso a servir.
  - La capacidad de la colas
  - La cantidad de las colas.
  - Las reglas de encolado y descarte.

La relación entre la velocidad de llegada y la tasa de servicio dará el tiempo que los paquetes deben esperar en el sistema. Cuando un paquete llega al sistema y el despachador está ocupado sirviendo a otro, se incrementará el tiempo del paquete dentro del sistema. La cantidad de tiempo que el proceso debe esperar depende de las características del sistema de encolado y de los factores externos. Debido a que los routers tienen una cantidad limitada de recursos (memoria intermedia y ancho de banda del puerto de salida, etc) las colas serán finitas en cuanto a la longitud. Si se alcanza la capacidad máxima de la cola (se llena) los paquetes nuevos que llegan serán descartados, aunque esto no es siempre así, dependerá de las reglas que regulan

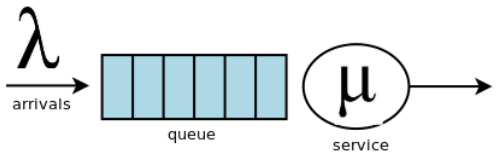


Figura 4.1: Diagrama de un modelo de cola M/M/1

el sistema: **Queueing Rules**.

Habitualmente se considera como sistema base para el estudio de colas un modelo $M/M/1$, el cual se ilustra en la figura 4.1. Este es un sistema que en la mayoría de los casos es difícil de aplicar al tráfico de paquetes de datos, ya que supone que el proceso de llegada se comporta bajo la hipótesis de Poisson. Para más información se puede consultar la bibliografía relacionada, como [Sch94] [Bol98].

La hipótesis de Poisson no siempre es cierta, y tal vez nunca suceda, por ejemplo ante tráfico con comportamiento autosimilar. Como normas generales podemos mencionar que si hay aumento del tráfico, el retraso aumentará, y en presencia de tráfico en ráfagas, el retardo será más variable, dando peores resultados. Si la carga aumenta se excederá la capacidad del router dando una mayor longitud de la cola pudiendo llegar al estado irreal de longitud infinita. En un sistema real el router se verá obligado a descartar los paquetes. Una solución podría ser adquirir un equipo más potente y aumentar el ancho de banda de la salida. A menudo el **overprovisioning** (sobre aprovisionamiento), situación donde siempre sobran los recursos, no es posible. Al corto plazo los routers deben adaptarse a los cambios y en lo posible hacerlo de una forma "inteligente" tratando de reducir los efectos secundarios.

4.3. Colas en el router

Los routers tienen al menos dos niveles de colas, llamadas **colas de hardware** y **colas de software**. Se puede encontrar un tercer tipo, colas de bajo nivel, que incluye las **colas de interfaces de red**, las cuales residen en el chipset, por ejemplo, en las NIC Ethernet, y a menudo no son más que un pequeño banco de registros. Las colas mencionadas como hardware y software residen en el "espacio del kernel". Las colas de las NIC están relacionadas directamente con el diseño del hardware y puede variar entre cada fabricante.

A pesar de su nombre, las colas de hardware no están en los dispositivos, se encuentran entre el controlador de la interfaz y el kernel del sistema, en particular en el sub-sistema de I/O del sistema operativo y no, como su nombre lo sugiere, en registros físicos de la placa de red. Las colas llamadas de hardware son el primer nivel entre la red y el sistema, independientemente del hardware instalado. Se las conoce como RX-ring y TX-ring. Reciben este nombre porque casi siempre son implementadas con una estructura de descriptores de buffers de memoria que se disponen lógicamente en un arreglo FIFO, comúnmente no están dispuestos en bloques continuos de memoria, pero sí vinculados en una estructura de datos de lista doblemente enlazada y circular. Los controladores de las interfaces de red tienen acceso directo a estos espacios de memoria. Existe un par por cada NIC: RX-ring y TX-ring. En la figura 4.2 se muestra un esquema de las colas dentro de un router.

En los sistemas operativos basados en BSD Unix la misma estructura de datos se utiliza a lo largo de todo el stack TCP/IP. Los trozos de memoria se nombran `mbufs` y se reservan en tamaños pequeños. La estructura puede variar entre las diferentes implementaciones, incluso llamarse de forma distinta. Por ejemplo, los buffers en GNU/Linux se llaman `sk_buffs` y los datos se reservan en grandes bloques lineales, desperdiciando algunos bytes pero permitiendo un acceso más rápido. Las operaciones sobre los `mbufs` pueden hacerse de forma "clusterizada", es decir, tratadas en grupos.

Para los frames de entrada al sistema, el tratamiento se lleva a cabo de manera asíncrona. Cuando la tarjeta de red ha recibido una trama completa y correcta (tamaño y CRC están bien) se debe invocar al controlador específico del protocolo de la capa de enlace. Se puede realizar la tarea mediante un sondeo de la cola de la NIC, user polling, o esperar hasta que se interrumpa el sistema a través de una IRQ, INTR de hardware. Los mecanismos de interrupción son más apropiados, aunque los detalles dependerán de la implementación, como son los drivers y la arquitectura de hardware. El sistema podrá mantener la información relacionada al buffer del dispositivo en la memoria de E/S, memoria principal, memoria DMA o registros específicos. Para las colas de entrada las implementaciones de hardware son suficientes.

Cuando el frame entrante llega al sistema entra en los registros de alguna de las interfaces (cola de la NIC) y luego se maneja inmediatamente por el controlador del dispositivo (una parte de software que comunica al sistema operativo con el hardware). Una vez que se invoca el controlador, este tomará el frame de la cola de la

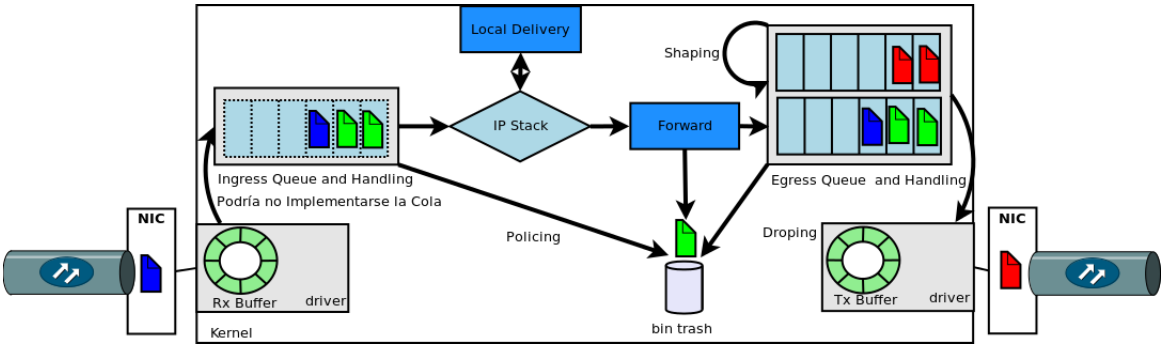


Figura 4.2: Diagrama de los sistemas de colas manejados por un router

tarjeta de red y lo colocará en un buffer del sistema operativo (RX-ring, conocido como cola de hardware). Para acelerar esta tarea, estos movimientos de datos, desde medios físicos a la memoria reservada del kernel, se pueden realizar mediante acceso directo a memoria o con tecnologías de buses modernas, que terminan asemejándose más a un switch interno que a un bus.

Después, el encabezado y el trailer (cola) de la trama se comprueban (CRC y tamaño se calculan generalmente por el hardware, en la interfaz) se podrán finalmente entregar los datos a un protocolo de capa 2 ó 3 (ARP, STP, IPv4, IPv6, ...) particular que continuará su procesamiento.

Un enfoque muy utilizado es mantener, por el sistema operativo, una cola de hardware para cada protocolo que trabaje sobre la capa de enlace de datos que implementa la NIC. El primer nivel de multiplexación se realiza mediante la rutina de la capa de enlace, si la carga útil de la trama es, por ejemplo, IPv6 de acuerdo al campo Ethertype, el controlador de protocolo determinado recibirá una interrupción de software (esto dependerá de la implementación del stack TCP/IP) y el proceso de reenvío, que se ejecuta en el CPU principal del router o en hardware dedicado ASIC, procesará el paquete de su buffer.

Los routers tienen colas de entrada y de salida. La cola de entrada estará la mayor parte del tiempo vacía. Esto sucede porque el reenvío/forwarding es un proceso muy rápido que depende de la CPU del router que siempre es mucho más rápida que la velocidad de la interfaz de red). Rara vez se requerirá buffering, aunque puede ser necesario debido a un alto tráfico de entrada. En esta situación se utiliza una cola de

entrada de software. Otra posible implementación podría elegir descartar el paquete en lugar de ponerlo en la cola de software.

Para las colas de salida la situación es un tanto diferente, por lo general estarán la mayor parte del tiempo ocupadas. La velocidad de los datos entrantes, la capacidad de router y el ancho de banda digital de la interfaz de salida determinarán el estado de las colas. El proceso de forwarding tiene que manejar el paquete, eligiendo la interfaz de salida y finalmente, colocar el mensaje en el buffer de transmisión, TX-ring de la interfaz seleccionada. Una vez que los paquetes se colocan en el TX-ring, su posición no puede ser cambiada. El controlador del dispositivo debe ahora copiar los datos en los registros de interfaz e iniciar el proceso de transmisión física. Una vez que los datos se copian en la memoria de la tarjeta de red, las posiciones usadas en las colas de hardware pueden ser liberadas.

Sólo cuando el TX-ring está lleno la cola de software debe encargarse de manejar los paquetes. Por lo tanto, la configuración de colas de software sólo tiene efecto durante períodos de congestión de la interfaz, cuando se desbordó la cola de hardware. La longitud del TX-ring, al igual que la del RX-ring, se configura automáticamente por el sistema y depende de la velocidad de la interfaz. Las colas de hardware tienden a no ser demasiado largas, de manera de no introducir mucho delay. Para los enlaces de baja velocidad (menos de 3Mbps) sólo tienen 2 o 4 lugares por frame. Para los enlaces de alta velocidad, como una conexión Ethernet de 100Mbps, su tamaño puede ser de hasta 128 entradas. Su longitud podría ser configurada manualmente, pero no es algo recomendado. Pocas veces se requiere cambiar los ajustes por default del sistema para el parámetro. Un escenario posible sería en el caso que el tamaño de la cola de hardware es tan larga que produce retrasos inaceptables reteniendo el tráfico sensible al delay.

El TX-ring se descarga tan pronto la interfaz puede despachar el siguiente paquete. Esto se realiza sin necesidad de interrupciones. Los routers deben proveer implementaciones para que los buffers de bajo nivel tengan acceso al TX-ring de forma directa sin involucrar al CPU de propósito general.

Como se mencionó, el RX-ring está generalmente vacío y sólo de vez en cuando se ve cargado. En cambio, el TX-ring se ve desbordado con frecuencia. Cuando el TX-ring está lleno, las colas de software entrar en juego. Para la planificación de las colas

de salida por software existe una variedad de **Disciplinas de encolado (Queueing Disciplines)**. Una disciplina es un forma de trabajar, una programación de la cola mediante la cual se indica la forma en que los paquetes, se agregan, se quitan para ser procesados o se descartan. Define las reglas de encolado, desencolado y descarte sobre la misma.

4.4. Disciplinas de encolado

4.4.1. Encolado FIFO/FCFS

First-In, First Out (FIFO) o First-Come, First-Served (FCFS) es el algoritmo más simple. Los paquetes son atendidos y enrutados a través del sistema enviándose por la correspondiente interfaz de salida de acuerdo al orden en que llegaron. Todos los paquetes son tratados por igual. Cuando un paquete llega, y el sistema (router) está ocupado, se coloca en una cola de salida única (espacio de buffer). Dado que la cantidad de memoria en cada router es finita, el espacio se podría llenar. Si no hay suficiente espacio los paquetes deberán, de acuerdo a la política de drop, ser descartados. En el caso FIFO la política es muy sencilla, los paquetes nuevos que arriban cuando la cola está llena se descartan y los más viejos permanecerán esperando a los que se encolaron previamente. Esta política de descarte se conoce como **Tail Drop**. Cuando es el turno para el paquete, se quita de la cola, se procesa y, finalmente, se envía a través de una interfaz de salida (si el router encuentra con éxito una ruta).

Los colas FIFO son el mecanismo por default utilizado en los dispositivos de red para procesar paquetes. La mayoría de los routers de Internet funcionaban de una manera FIFO estricta. Se trata de una solución básica, pero que no proporciona mecanismos para QoS, todos los paquetes obtendrán la misma manipulación por lo que los retardos pueden variar en función de la carga del router. Este manejo no permite a los dispositivos de red organizar, reordenar los paquetes en forma diferente, por lo tanto, no pueden dar un servicio a un flujo de datos particular de manera diferente que el resto del tráfico.

En ocasiones un flujo "brusty" en una cola FIFO permite que se consuma todo el espacio en memoria de la misma, haciendo que todos los otros flujos (tal vez de buen comportamiento) no puedan recibir un servicio adecuado. El llenado de la cola produce un bloqueo durante un intervalo de tiempo. El tráfico TCP es más susceptible

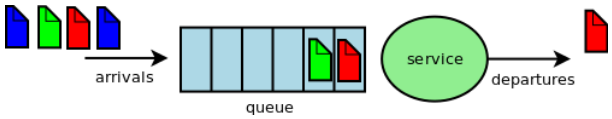


Figura 4.3: Diagrama de una cola con política FIFO

a sufrir una negación temporal de servicio. Gráficamente el modelo FIFO es descripto por la figura 4.3. Las colas FIFO puede ofrecer los siguientes beneficios:

- Carga computacional baja.
- Tiene un comportamiento muy predecible para los paquetes, aunque no bajo carga. El retardo máximo puede ser determinado por la longitud máxima de la cola.
- Las colas FIFO son adecuados para sistemas que siempre funciona sin carga extrema o tienen interfaces muy rápidas.

Las limitación más importante es la incapacidad de proporcionar un planificador de QoS durante períodos de congestión.

4.4.2. Priority Queueing (PQ)

Las colas de prioridad (PQ) es un método relativamente simple para soportar clases diferenciadas de servicio. En lugar de utilizar una sola cola, este sistema aumenta el número de colas y le asigna un valor de prioridad a cada una. La idea es priorizar el tráfico entre las diferentes colas.

PQ es una técnica que requiere un proceso de clasificación antes de la puesta en cola de los paquetes. Los paquetes que llegan deben ser clasificados en las clases de prioridad. Cada clase tiene una prioridad y está asociada a una cola. Los paquetes se colocan en la cola correspondiente de acuerdo con su clase. Las colas se procesan en orden, desde la de mayor prioridad a la más baja. Cuando el router elige un paquete para transmitirlo procesará el tráfico de cola de más alto valor. La tarea de procesamiento de tráfico permanecerá en la cola de prioridad más alta hasta que esta se vacíe. No empezará a procesar la siguiente cola (en orden de prioridad estricta) mientras existan paquetes de tráfico de mayor prioridad.

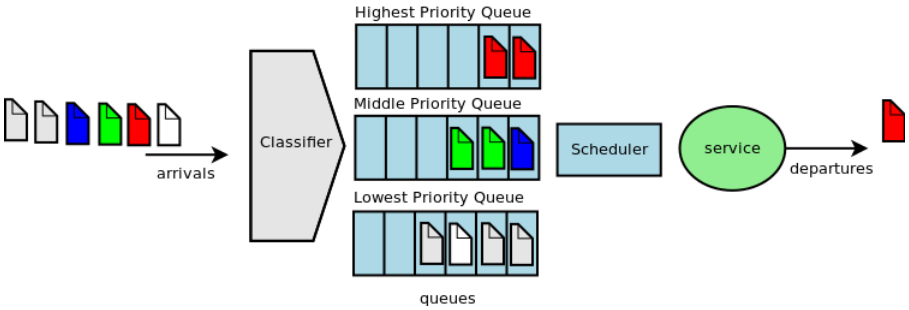


Figura 4.4: Diagrama de una cola con política PQ

Este algoritmo se implementa con frecuencia de una manera no apropiativa (non-preemptive), la transmisión de un paquete no se interrumpe una vez que comenzó. Recién cuando se envió, el sistema re-inspeccionara todas las colas y seleccionara la de mayor prioridad que no esté vacía. Cada cola individual se procesa con el algoritmo FIFO visto anteriormente.

PQ está diseñado para dar soporte a las aplicaciones de misión crítica, como los programas de tiempo real duro. Un problema de este método de procesamiento es que los paquetes pueden sufrir de inanición (starvation) si una cola de alta prioridad nunca se vacía. La inanición es el principal inconveniente de las colas PQ.

El proceso de clasificación previo, antes del encolado, se lleva a cabo buscando campos específicos de los datagramas IP, segmentos TCP y/o datagramas UDP.

PQ es la base para los algoritmos de planificación de cola diseñados para proporcionar soporte a las clases de servicio diferenciadas, pero su procesamiento estricto puede derivar en una red donde un tipo de tráfico nunca sea procesado. Las implementaciones suelen definir un número finito, y bajo, de colas (entre 4 a 6). La asignación del tráfico a las colas se realiza de forma estática por reglas que coinciden con la información de la cabecera de los paquetes, como los campos direcciones, protocolo o los números de puertos. Siempre debe existir una cola de baja prioridad para todo el tráfico no coincidente y es necesario que el sistema se reserve una cola para el tráfico del plano de control. Debido al bajo número de colas, puede considerarse una solución de grano grueso (coarse-grained). Esta técnica de encolado es ilustrada en la figura 4.4. PQ puede ofrecer los siguientes beneficios:

- Baja carga computacional.

- Permite mejorar la estabilidad de la red en periodos de congestión si los protocolos del plano de control se consideran como los más prioritarios.
- Soporta aplicaciones de tiempo real como emulación de circuitos TDM.

Es crucial que el tráfico que entra al dominio para luego tratarse con este sistema, sea previamente acondicionado, es decir, remarcado de acuerdo a su clase. Se recomienda su aplicación sobre servicios que no son burstiness y que tienen características de CBR (constant bit rate) o su comportamiento es bien conocido, tales como emulación de circuitos TDM o VoIP, donde se puede conocer el tamaño del paquete, el volumen de tráfico, y el comportamiento del mismo.

4.4.3. Fair Queueing (FQ)

El objetivo de la disciplina Fair Queueing es permitir que los datos de múltiples flujos puedan compartir equitativamente el ancho de banda digital disponible en un enlace. El router a implementar este método debe mantener una cola independiente para cada flujo detectado. Todas estas colas se deben atender de una manera Round Robin (RR): un paquete por vez de cada cola, por lo tanto el router asegura que cada flujo tenga la misma posibilidad de ser atendido. Al momento de tomar los paquetes de las colas, aquellas vacías son salteadas y pierden su turno. La idea detrás de este método es compartir de forma equitativa, en inglés "Fairly Sharing".

Debido a que los paquetes varían en tamaño, cuando el sistema de cola los procesa, puede encontrar que se excede la cuota otorgada en su turno. Una posible implementación es que el sistema envíe el paquete completo pero penalicé la cola para la próxima pasada en la suma que se excedió.

Esta técnica requiere la detección de flujos y un proceso bastante costoso de clasificación antes de que los paquetes sean encolados. FQ resuelve el problema del "bloqueo temporal" del método FIFO y la inanición posible de PQ, pero el problema que surge es que todos los flujos son tratados de la misma manera, por lo tanto, no existe la posibilidad de proporcionar un tratamiento diferencial para aquellos paquetes de flujos que así lo requieran. Este enfoque es considerado una solución de grano fino (fine-grained) debido la "granularidad" generada para los diferentes flujos. Las implementaciones habituales a menudo utilizan un número de colas de más de 200. Debido a la auto-configuración se recomienda para sistemas de baja velocidad; como líneas de

datos T1, E1, Frame-Relay; o cualquier línea digital de menos de 3 Mbps de capacidad.

Por supuesto, el espacio de memoria para los buffers es finito, por lo que los tamaños y cantidad de colas son finitos también. El espacio de memoria intermedia se divide en N colas, cada una de los cuales se utiliza para mantener los paquetes de un flujo. La detección de flujo se puede lograr con las técnicas vistas en el capítulo que trata IntServ. Una posible aplicación podría ser utilizar una función hash sobre el campos significativos de paquetes:

- IP Address origen.
- IP Address destino.
- Protocolo o next-header.
- Port origen.
- Port destino.
- IP ToS/DSCP/Precedence (QoS markers).
- Flow label.

Si hay más flujos que colas los flujos deberán compartir colas. El llenado de todas las colas podría ocurrir por lo que se deberá aplicar una política de descarte.

El método de FQ fue propuesto por John Nagle en 1985 en el RFC-970. Este es la base para un grupo de técnicas de encolados diseñadas para evitar que algunos flujos puedan monopolizar el uso de los recursos de un router. FQ también puede ser denominado con el término Flowbased Queueing (FBQ).

Con este método aplicado sobre un enlace de datos con un ancho de banda digital de B y donde hay N colas activas, cada flujo obtendrá una tasa de datos promedio de valor B/N. Los flujos con "mal comportamiento" sólo castigarán a los paquetes de su clase, así otras sesiones no se verán afectadas por los flujos que no se regulan. Un diagrama de este sistema es ilustrado por la figura 4.5.

FQ puede ofrecer los siguientes beneficios:

- Compartir equitativamente el ancho de banda digital.
- Los bursty flows no degradarán el tratamiento para otros flujos.

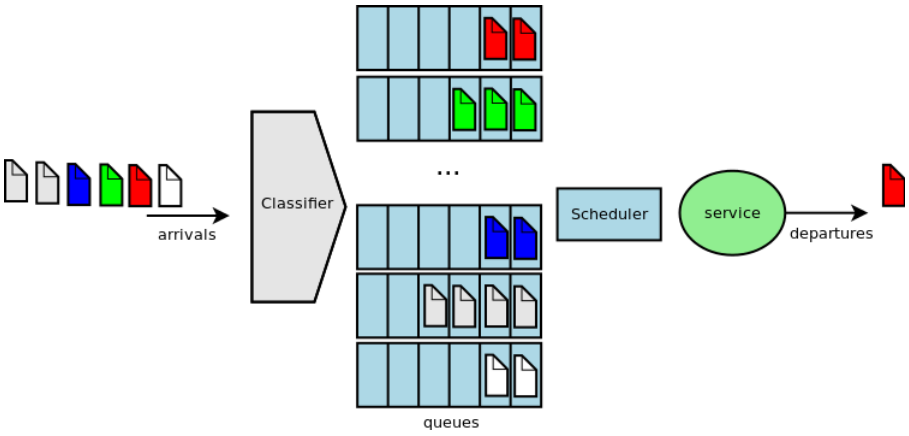


Figura 4.5: Diagrama de una cola con política FQ

Las limitaciones son, como se mencionó, que FQ no proporciona un mecanismo que le permita soportar diferentes niveles de QoS como para distinguir los servicios en tiempo real de otros flujos. La clasificación no es una tarea fácil y no se puede aplicar en los routers de core, porque ellos reciben miles o decenas de miles de flujos, lo que impactaría negativamente en el rendimiento y la escalabilidad. FQ supone que el tráfico se puede separar en flujos bien definidos, pero esto no siempre es cierto. FQ se aplica típicamente en los bordes de la red, "contra" los clientes, en donde ellos se conectan al proveedor de servicios.

4.4.4. Fair Queueing Estocástico (SFQ)

Stochastic Fairness Queuing (SFQ) es considerado miembro de la familia de las disciplinas de encolamiento basado en FQ. Esta es una versión estadística del algoritmo FQ. Trata de resolver el problema que surge cuando hay demasiados flujos y las tablas de los mismos son muy grandes para ser procesadas con FQ puro. Se resuelve el problema de la gestión colocando sólo una pequeña cantidad de colas y haciendo una correlación aleatoria entre los flujos de una cola. Se requiere menos cálculos mientras que resulta ser "casi" justo. SFQ se llama " estocástico" porque en realidad no se asigna una cola para cada sesión, el algoritmo divide el tráfico sobre un número limitado de colas usando una función de hash. Con este método varias sesiones terminarán en la misma cola. Para reducir que siempre las mismas sesiones compartan la misma cola puede ir variando el algoritmo de hash, de forma que si comparten, lo harán solo por un tiempo limitado.

4.4.5. Weighted Fair Queueing (WFQ)

Weighted Fair Queueing (WFQ) es una disciplina desarrollada por Lixia Zhang [Zha90]. Otras propuestas fueron presentadas simultáneamente por Alan Demers, Srivinasan Keshav y Scott Shenke en 1989. La idea detrás del método es similar a FQ, proveer "equidad entre flujos", pero con capacidad para proporcionar servicios diferenciados para algunos. Weighted Fair Queueing (WFQ) puede ser visto como una generalización de FQ. En la planificación de FQ la cantidad de ancho de banda se divide proporcionalmente en la cantidad de colas. Esto conduce a que todos los flujos sean atendidos por igual, así no hay un tratamiento especial. WFQ le da además a cada cola diferentes acciones de servicio. Cada cola tiene asignado un valor de peso, esto es directamente proporcional con la cantidad de paquetes que podrá procesar por ciclo de Round Robin la cola. Por lo tanto, un mayor peso da una mayor prioridad. A diferencia de FQ, donde se atienden las colas en forma de Round Robin, en WFQ se atienden las colas con Weighted Round Robin (WRR). El valor de peso para cada flujo se calcula a partir, por ejemplo, del campo ToS/DSCP/Precedencia/Traffic Class de los datagramas IP.

WFQ es capaz de manejar el tráfico interactivo (de bajo volumen) de forma que fluya con un mejor tratamiento y así, reducir el tiempo de respuesta y luego compartir el ancho de banda restante entre los demás flujos de datos.

WFQ, de la misma forma que FQ, requiere categorizar el tráfico de acuerdo a los diferentes flujos. El proceso es clave para poder ofrecer un tratamiento diferenciado. Para obtener buenos resultados el marcado de los paquetes, que luego serán clasificados, debe ser confiable. Además de los campos, ToS/Precedence/DSCP/Traffic Class, se puede inspeccionar en la clasificación el campo **Flow Label** en IPv6 o incluso el campo **Class of Service** que tienen los protocolos a nivel de enlace, como por ejemplo, MPLS o 802.1p con Ethernet.

Los paquetes arriban, son clasificados mediante la inspección de los campos o de acuerdo a la interfaz por la cual ingresan y se mapean a partir de una función de hash a una cola determinada donde esperarán hasta ser enviados. La generación de las colas y los pesos asignados se realiza de forma automática a través de métodos aún en desarrollo, llamados a menudo "state-of-the art".

Los beneficios que puede ofrecer la disciplina WFQ son:

- Compartir equitativamente el ancho de banda digital.
- Los bursty flows no degradarán el tratamiento para otros flujos.
- Se puede dar un tratamiento diferenciado a los flujos más prioritarios.

Como contra el mecanismo de WFQ es difícil de implementar en hardware y los usos actuales son vistos solo en componentes de software. Tiene básicamente las mismas limitaciones que FQ. Al ser complejo suele requerir uso intensivo de CPU y memoria. Rara vez se lo ve en equipos de core. Algunas implementaciones que intentas resolver sus limitaciones son Self-clocking Fair Queuing (SCFQ), Worst-case Fair Weighted Fair Queuing (WF2Q) [BZ96] y Worst-case Fair Weighted Fair Queuing plus (WF2Q+).

4.4.6. Class Based / Weighted Round Robin Queueing

Las disciplinas Class Based Queueing (CBQ), Weighted Round Robin (WRRQ) y Custom Queueing (CQ) son sistemas de colas que dividen el ancho de banda digital disponible estadísticamente entre las múltiples clases definidas por el usuario en una modalidad de grano grueso. La idea detrás de estos métodos es la de "Garantizar un ancho de banda" a cada tráfico. Cada clase tendrá su cola asociada y los paquetes serán encolados en la correspondiente de acuerdo a una inspección previa de los campos o de marcas previas que traigan.

La asignación entre los paquetes y la cola se pude configurar de forma manual, como sucede con un sistema PQ. CBQ/WRRQ utilizan los mismos mecanismos de clasificación. Como se manejan pesos, las colas no tienen el mismo tratamiento. Existen colas que obtendrán un mayor porcentaje del total del ancho de banda disponible, siendo de esta forma priorizadas.

CBQ/WRRQ resuelve el problema del tratado estricto que tiene PQ, permitiendo al tráfico de menor prioridad ser servido y simplifica la clasificación con respecto al sistema WFQ en la fase previa, ya que se desarrolla una solución sin una gran cantidad de colas, como se indicó, se utiliza una metodología de grano grueso. Como contra tiene que la puesta en producción suele ser más compleja debido a que requiere una configuración, a diferencia de WFQ que puede funcionar de forma casi automática.

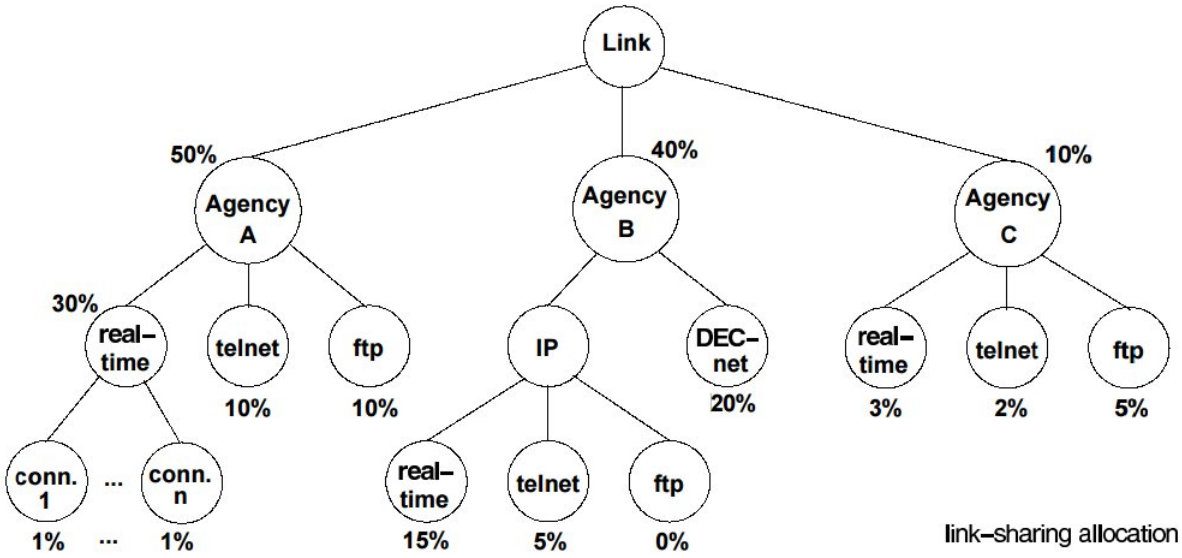


Figura 4.6: Diagrama jerárquico de una cola CBQ

CBQ (Class Based Queueing) fue inicialmente definido en los artículos de D. Clark y V. Jacobson, "Flexible and Efficient Resource Management for Datagram Networks" de 1991 o el de Sally Floyd "Link-sharing and Resource Management Models for Packet Networks" [FJ95]. En dicho sistema, las colas son organizadas de forma jerárquica. Un ejemplo gráfico se puede observar en la figura 4.6 tomada del artículo indicado previamente.

En el tope de la jerarquía existe una cola tipo "catch-all", llamada raíz, ROOT. La raíz tiene disponible una cantidad de ancho de banda del total. Luego, las colas hijas de ROOT, tendrán asignada una porción del total que tiene la cola raíz. En ocasiones el ancho de banda sobrante que no se usa por algunas colas hijas se puede usar por las colas hermanas que lo necesitan. Este esquema se puede repetir de forma recursiva generando una ramificación a partir de otra anterior. En cada nivel de colas la suma del ancho de banda asignado no puede ser mayor que el total de la cola madre, raíz. En el sistema se debe considerar una cola especial llamada "Cola de Sistema" para los servicios críticos del plano de control.

El sistema Custom Queuing (CQ) es un ejemplo de una implementación de WRR. El término general es WRR. CQ es similar a CBQ, pero no construye una estructura jerárquica. CQ, como CBQ, reserva un porcentaje del total para los diferentes tráficos que serán ubicado en sus determinadas colas. Solo hay un nivel de colas en este

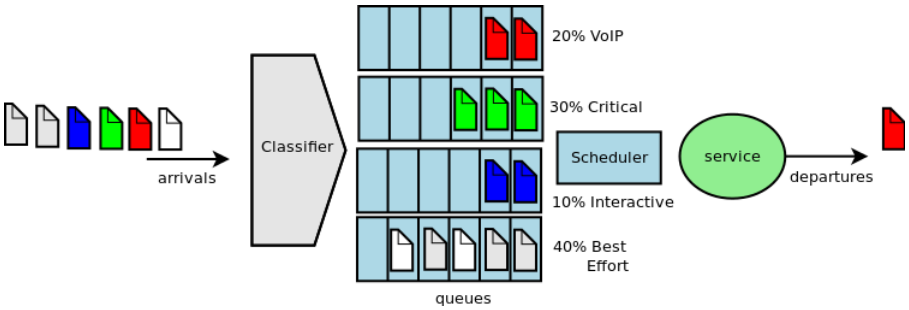


Figura 4.7: Diagrama de una cola con política CQ

sistema. Si una cola no usa determinado ancho de banda asignado, este puede ser utilizada por las demás si lo requieren. CQ es el nombre que el fabricante cisco le da al sistema de colas definidas por el usuario con prioridad sin inanición. Otros fabricantes como juniper directamente las llama WRR. La figura 4.7 sirve para representar un diagrama de una cola WRR o CQ. Las colas CBQ/WRRQ/CQ proveen los siguientes beneficios:

- El sistema de colas puede ser implementado en hardware dedicado por lo tanto se puede aplicar a interfaces de alta velocidad, tanto en el núcleo como en los bordes de la red.
- Proveen una solución de grano grueso y dividen el total del ancho de banda digital en diferentes porcentajes de acuerdo a la prioridad del servicio.
- Pueden ser usadas en sistemas de tiempo real compartiendo con tráfico best-effort, donde no se producirá inanición para este último.

La mayor limitación de los sistemas CBQ/WRRQ es que son modelos teóricos que permiten dividir el ancho de banda de acuerdo a lo indicado por el usuario de forma equitativa si los promedios de los tamaños de los paquetes son similares, pero si los tamaños de algún tipo de tráfico es más grande que el resto en promedio podrá obtener mayor ventaja. Otra limitación como sucede con PQ, es que, al ser definidas de forma manual no se adaptarán de forma dinámica a las condiciones de la red.

4.4.7. Deficit Weighted Round Robin (DWRR)

El sistema de colas Deficit Weighted Round Robin (DWRR) propuesto por M. Shreedhar y G. Varghese en 1995 fue diseñado para intentar resolver algunos problemas de implementación de WRRQ/CBQ. En realidad, los sistemas en funcionamiento

llamados WRRQ y CBQ se basan en esta implementación de DWRR. DWRR trabaja con un planificado que requiere menos recursos computacionales que CBQ o WRRQ puros. En el nuevo sistema cada cola tiene asignado un porcentaje del ancho de banda asociado a un contador llamado **DeficitCounter**. En el contador se indica la cantidad de bytes que tiene la cola permitido transmitir. Este valor es actualizado en los quantum del servicio en el que el tiempo es dividido. O sea, se incrementa conforme pasa el tiempo, como sucede con los tokens en un token bucket. Cada vez que le toca a la cola ser procesada, el planificador verifica el valor del contador, si el paquete tiene la cantidad de bytes permitidos lo procesa y decrementa el contador correspondiente la suma de bytes igual al paquete despachado, luego continuará procesando la misma cola hasta llegar a un valor que no permita despachar el paquete en curso (un valor menor que la cantidad de bytes del paquete), ahí pasara a analizar la siguiente cola.

4.4.8. Class-Based Weighted Fair Queueing (CBWFQ)

Class-based weighted fair queueing (CBWFQ) es un sistema de colas que extiende la funcionalidad de WFQ para proveer colas definidas por el usuario como CQ o CBQ. Es una combinación entre colas definidas por el usuario y colas dinámicas generadas a partir de los flujos detectados.

Cada clase se define a partir de reglas que hace "matching" con determinado tráfico, luego se asigna un ancho de banda, un peso y un limite máximo de paquetes encolados para la misma. El ancho de banda digital indicado debe ser garantizado para la clase durante los períodos de congestión. Cuando la cola se llena, las políticas de descarte se deben aplicar. Se puede utilizar una de tail drop o descartes por prioridad antes de producirse el llenado completo de la cola. Tail drop es el mecanismo default. Se puede aplicar un descarte Weighted Random Early Detection (WRED) antes de que la cola se complete indicando los límites. En este sistema de colas se puede configurar una clase default o catch-all, que puede ser una mecanismo WFQ, de esta forma se combinan colas de usuario con un WFQ. El nombre CBWFQ es dado por cisco a una implementación dada. En la figura 4.8 se muestra un diagrama de una cola con esta política. En GNU/Linux el sistema CBWFQ es más engorroso de configurar, por lo que se ofrece como alternativa el sistema **Hierarchical Token Bucket (HTB)** el cual se muestra en al apéndice correspondiente.

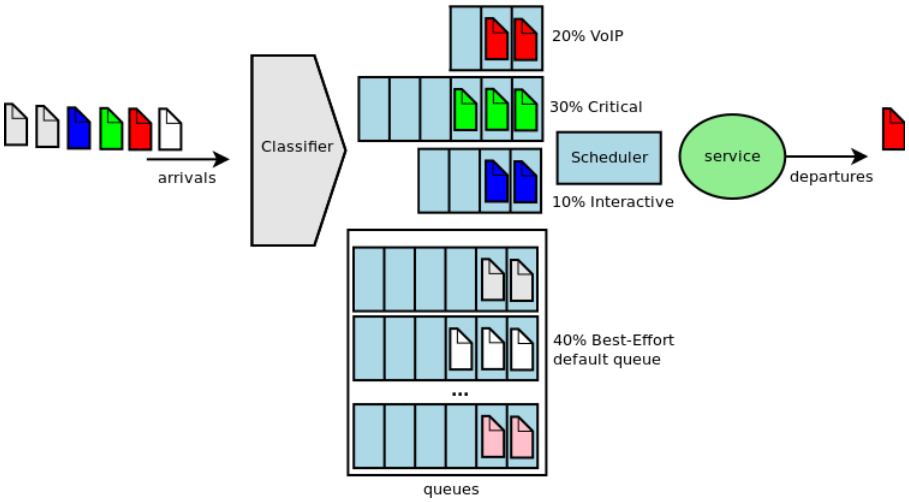


Figura 4.8: Diagrama de una cola con política CBWFQ

4.4.9. LLQ

Otro tipo de sistema de colas implementado en routers cisco es el llamado Low Latency Queuing (LLQ). Este provee un manejo de prioridades estrictas, como PQ, al modelo de CBWFQ. CBWFQ carece de un mecanismo para priorizar tráfico sensitivo al delay y tratarlo en un orden estricto. LLQ agrega esta característica, pero, para no producir una posible inanición como sucede con PQ, solo servirá este tráfico de forma prioritaria en un ancho de banda fijo (limitado). Una vez utilizada su parte, el resto de los paquetes de la clase serán descartados, dando lugar así al procesamiento de las demás colas en forma CBWFQ.

Este modelo se implementa sobre CBWFQ indicando luego cual de las colas será donde se colocará el tráfico sensitivo al delay, dando prioridad estricta en el manejo de los paquetes de esta, siempre dentro del ancho de banda establecido. Los paquetes que no están dentro de los límites serán policed mediante un descarte.

4.5. Técnicas para descarte de datagramas

En la siguiente sección del texto se muestran diferentes técnicas de descarte de paquetes. Estas acompañan las políticas de manejo de tráfico para trabaja con los sistemas de colas.

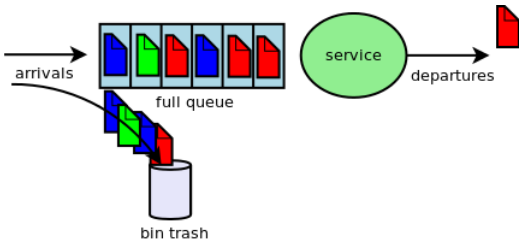


Figura 4.9: Descarte clásico Tail Drop

4.5.1. Técnica de descarte Tail Drop

Durante los períodos de congestión las colas llenas no tienen más capacidad para recibir nuevos paquetes y el planificador se ve forzado a aplicar algoritmos de descarte. El más común es el llamado descarte desde los últimos de la cola, o en inglés **Tail Drop** o **Drop Tail**, en el que, simplemente los que llegan "tarde", una vez que la cola se llenó no serán recibidos, por lo tanto, se pierden. Esto se aplica hasta que exista nuevamente lugar en la estructura de datos. En la figura 4.9 se ilustra esta técnica de descarte.

Acorde a los protocolos transportados será la reacción ante el descarte. Por ejemplo, para TCP causará que la sesión entre a la fase de slow-start (SS) reduciendo drásticamente el uso que puede hacer del enlace. Para UDP la reacción dependerá de la capa superior. Comúnmente UDP se mantiene sin cambios en su tasa de transmisión ante la pérdida de datagramas.

Algunos estudios han mostrado que la técnica de descarte de múltiples segmentos de conexiones TCP diferentes en el mismo equipo por la falta de espacio en la cola genera una situación conocida como **TCP global synchronization** [Has89]. En este caso, todos los emisores TCP afectados entrarán en el control de congestión "reseteando" la variable CWD (congestion window) y volviendo a comenzar casi al mismo tiempo, dando como resultado otro evento de congestión en el corto plazo produciendo nuevamente el mismo efecto. Esto se puede ver como un incremento y un decremento "casi" global del uso del ancho de banda produciendo una sub-utilización de la red. La figura 4.10 intenta ilustrar esta situación tomando como fuente varias sesiones TCP.

Se remarca en otros estudios que el problema se debe al sencillo mecanismo de descarte del final cuando la cola está llena. En el artículo se recomienda aplicar algoritmos de descarte más inteligentes para manejar las colas. Se resalta que el

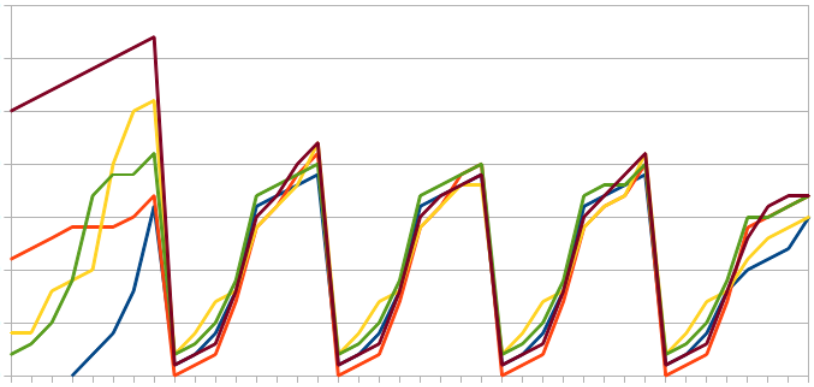


Figura 4.10: Diagrama del problema de la sincronización global de TCP

efecto se produce donde flujos, probablemente sin ninguna relación alguna, se vinculan sincronizándose por el hecho de ser descartados al mismo tiempo, volviendo más tarde a encontrarse en la misma situación del estado de la cola llena.

4.5.2. Técnica de descarte Drop-from-Front

Sobre una cola completa, esta técnica, en lugar de descartar los que arriban fuera del espacio de memoria reservado, desecha los paquetes más viejos, los que están en el tope de la cola para hacer lugar a los nuevos que aún no se almacenaron. Con este método se puede resolver algunos problemas de bloqueo producido por la técnica de descarte de la cola en un sistema FIFO. Un problema que se encuentra en un sistema FIFO con descarte desde la cola se conoce como "Head-of-line blocking" o directamente HOL blocking (un diagrama del mismo se muestra en la figura 4.11 tomado de Wikipedia.). Esta situación se encuentra en dispositivos con muchas interfaces de entrada y muchas de salidas, donde los bloqueos pueden producirse no por falta de capacidad sino cuando varios paquetes desde colas diferentes de entrada tienen como destino una misma cola de salida, retrasando así otros que podrían despacharse por interfaces sin carga. Otra técnica que permite resolver el problema de HOL blocking es la de no manejar directamente colas de entrada y tener varias colas virtuales de salida por interfaz, técnica conocida como **Virtual Output Queues** o **Advanced Input Queues**. Se implementa colocando en las interfaces de entrada una cola virtual separada por cada interfaz de salida, de esta forma se evitan parte de los bloqueos.

Drop-from-Front fue inicialmente diseñado para mejorar el rendimiento de TCP sobre redes con altos valores del producto $B \times D$ (bandwidth-delay networks) y utilizado en TCP over ATM. En el caso de ATM esta técnica genera espacio para nuevas

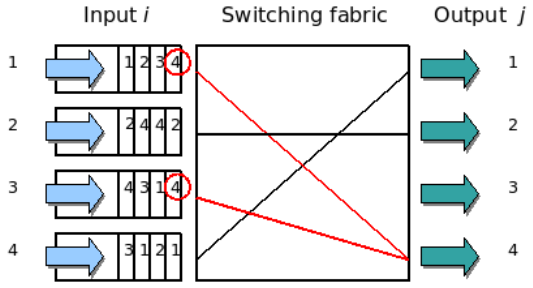


Figura 4.11: Problema de Head of Locking

celdas descartando las que están más cerca de la salida. TCP puede ser notificado del descarte de forma más temprana, previo a que el origen reaccione ante el timeout por no recibir la confirmación, ACK. Otras versiones como Partial Frame Drop at Front (PFDF), permite descartar celdas de forma selectiva pertenecientes al mismo paquete.

Las políticas o disciplinas de descarte Drop-tail o Drop-from-Front tienden a penalizar el tráfico en ráfagas y causan el problema de la sincronización global de TCP. Estudios posteriores demostraron que realizar un descarte probabilístico antes de que la cola este completa ayuda a resolver estas situaciones. AQM (Active Queue Management), Manejo activo de colas como disciplina aplica esta última técnica.

4.5.3. Técnica de descarte RED

La técnica RED (Random Early Detection] es un método usado en AQM, el cual trata de evitar el problema de la mala utilización de la red por el fenómeno llamado sincronización de TCP global. La técnica fue estudiada por varios investigadores en 1993, entre ellos Sally Floyd y Van Jacobson. Estos presentaron un artículo titulado "Random Early Detection Gateways for Congestion Avoidance" [FJ93], donde se brinda un borrador del algoritmo RED. Este no es solo una técnica de descarte sino también una manera de administrar la cola. Permite implementar el mecanismo de control de congestión evitando el problema de sincronización entre flujos TCP.

Un objetivo de esta técnica es proveer lo más pronto posible un feedback del estado de la red a los nodos que generan y reciben tráfico, antes que las colas "rebalsen" y desechen los paquetes. RED trabajará bien con flujos que tomen medidas ante las notificaciones de congestión, ya sea de forma explícita como con ECN, o de forma

implícita como lo hace TCP ante timeouts. RED se anticipa a la congestión indicando la situación antes que suceda.

Para aplicarse RED necesita controlar el promedio del tamaño de la cola y los paquetes marcados y/o descartados. Cuando la cola llega a un umbral, la selección de descarte o marcado comenzará a realizarse. Esta debe hacerse de forma equitativa sobre todos los flujos. RED será más eficiente en un entorno donde los extremos colaboran bajando las tasas de transmisión ante la inminente congestión, aunque también trabaja de forma adecuada sobre flujos que no toman medidas.

La primera tarea para evitar la situación extrema de descarte es detectar la incipiente congestión. Esto se realiza midiendo el tamaño de la cola durante períodos de tráfico o estimando durante los momentos que permanece vacía. Se puede computar un valor m que indica la cantidad de datagramas o paquetes que puede transmitir el dispositivo. RED no usa valores instantáneos sino que mide y calcula de forma estadística usando un filtro pasa-bajos, "suavizando" el promedio. Es computado como una función exponencial pesada, Exponential weighted moving average (EWMA).

$$avg_i = (1 - w_q)avg_{i-1} + w_q len(Q) \tag{4.1}$$

El peso w_q es una constante en el intervalo $(0, 1)$ con la cual se indica la relevancia de las mediciones más recientes del promedio avg. Si es grande se le da mayor relevancia y el tráfico en ráfagas dominara las decisiones. Si se pone muy bajo w_q, entonces las reacciones serán posiblemente muy lentas. El valor w_q dependerá del valor máximo posible de la longitud de la cola, L. y de dos umbrales: min_{th}, max_{th} que indican cuando comenzar a descartar y cuando detenerse.

Tres diferentes etapas se pueden encontrar en el arribo de un paquete:

- Cuando el promedio es menor que el valor min_{th}, en este caso no hay descarte ni marcado.
- Cuando el promedio está entre los dos umbrales, en este caso los paquetes serán descartados o marcados de forma aleatoria con cierta probabilidad, que se computará en función del promedio de la cola estimado.
- Cuando el promedio excede el segundo umbral, max_{th}, en este caso RED descartará o marcará cada paquete que llega al router.

La otra tarea es decidir a que flujos notificar de la congestión inminente mediante el descarte o marcado. RED lo hará de forma aleatoria cuando el promedio de la cola este entre los umbrales ya mencionados. Cada paquete que arriba será tomado en el proceso con una probabilidad p_a, donde este valor está en función del promedio, avg. El valor de p_a se calcula en base a otra probabilidad, p_b. Esta probabilidad se computa según la siguiente ecuación:

$$p_b = max_p(avg - min_{th})/(max_{th} - min_{th}). \tag{4.2}$$

Donde el parámetro max_p da el máximo para la probabilidad de marcado en p_b. Tanto como avg varía de min_{th} a max_{th} la probabilidad de marcado varía de forma lineal en el intervalo $(0..max_p)$. La probabilidad final p_a de marcado, descarte, es computada en base a p_b y se incrementa lentamente a medida que llegan paquetes desde el último marcado. Ahora si $p_a = p_b$ se obtiene una distribución geométrica de descartes entre llegadas, dando como resultado selección de ráfagas llevando al problema de la sincronización. Por eso $p_a \neq p_b$ sino:

$$p_a = \frac{p_b}{(1 - count \times p_b)} \tag{4.3}$$

Donde *count* es la cantidad de paquetes que llegaron desde el último que se marcó. Cada vez que se marca un paquete, p_a baja. En este caso p_a tiene una distribución uniforme.

RED puede medir la cola en bytes o paquetes. En caso que lo haga en bytes debe tener en cuenta que el paquete sea marcado en base a su tamaño en bytes.

$$p_b = max_p(avg - min_{th})/(max_{th} - min_{th}) \tag{4.4}$$

$$p_b = p_b \times \frac{PacketSize}{MaximumPacketSize} \tag{4.5}$$

$$p_a = \frac{p_b}{(1 - count \times p_b)} \tag{4.6}$$

Los paquetes más largos tendrán mayor probabilidad, beneficiando el tráfico interactivo, por ejemplo, mensajes VoIP, X-window, SSH o TELNET.

Haciendo un análisis del algoritmo se observan dos partes. Una que calcula el promedio de la cola y otra la probabilidad de marcado. La forma de evitar la congestión es tratando de mantener a la red en estado de baja latencia aprovechando el ancho de banda. La selección de los umbrales debe tener la característica tal que min_{th} permite

mantener una buena parte del tráfico que sea de ráfagas y puede ser computado como la máxima latencia de encolado permitida multiplicado por el ancho de banda digital total. El otro umbral, max_{th}, debe contemplar el máximo delay del router y debe ser lo suficientemente grande para prevenir la sincronización. Una regla habitual es colocarlo al doble del mínimo, $max_{th} = 2 \times min_{th}$.

RED aporta los siguientes beneficios:

- Control equitativo de la congestión.
- No hay sincronización global TCP.
- Se puede implementar en diferentes entornos.
- Aprovecha la red de forma global.
- Se puede implementar en casi cualquier router.

La desventaja más importante es que RED trata todos los flujos de forma equitativa sin tener en cuenta prioridades, no permitiendo así brindar una QoS diferenciada.

4.5.4. Técnica de descarte WRED

Weighted Random Early Detection (WRED) es un algoritmo desarrollado por cisco. Es una extensión de RED donde la misma cola puede tener varios umbrales, un par para cada tipo de tráfico a considerar. Cada par será (min_{th}, max_{th}) y estará asociado a un valor de IP Precedence, DSCP o Traffic Class. De esta forma se puede tratar diferente al tráfico de acuerdo al peso o prioridad. Cuando WRED descarta lo hará de forma selectiva de acuerdo a las prioridades indicadas. Esta técnica hoy puede ser encontrada en routers de otros fabricantes, como juniper. Este algoritmo puede ser usado en routers de core, donde ya el marcado del campo de Traffic Class fue realizado por los router de límite o borde al entrar al dominio de QoS.

GRED (Generic RED) es el nombre de WRED en los sistemas GNU/Linux. Los creadores son Werner Almesberger, Jamal Hadi Salim (el código C indica que el escritor del código es Jamal) y Alexey Kuznetsov. Este fue publicado en el artículo "Differentiated Service on Linux". En el caso de GRED se simula 16 colas virtuales con parámetros independientes.

Estos tres algoritmos, RED, WRED y GRED funcionan mejor cuando el tráfico es "responsivo" (responsiveness) a las notificaciones de congestión, como sucede con

TCP. RED no funcionará de forma óptima con protocolos que no reaccionan a estos avisos o descartes. Otro problema que se encuentra es que, de acuerdo a [MBDL99] la configuración de los parámetros adecuados es necesaria para un correcto funcionamiento, y si se dejan default probablemente no se tengan buenos resultados. Actualmente existen otras técnicas como ARED [FGS01] o **Blue** que permiten sintonizar los valores de forma automática, aunque no se han difundido las implementaciones en routers de la red para su uso en producción.

4.5.5. Weighted Tail Drop (WTD)

Weighted tail-drop (WTD) es una mejora sobre Tail Drop donde se tiene un programa mucho más sencillo que WRED o RED, donde se intenta descartar en base a prioridades, monitorizando el largo de la cola. Esta técnica se puede encontrar en dispositivos layer 2 usando el campo CoS (Class of Service) que proveen algunos protocolos de nivel de enlace.

4.5.6. Otras implementaciones

El "fabricante cisco ha desarrollado" otras tecnologías para evitar la congestión como puede ser **FbWRED (Flow-based Weighted Random Early Detection)**. Esta técnica se aplica a WRED, pero se hace por flujos y no por los valores de los campos DSCP o Traffic Class. La clasificación es bastante más compleja y en los routers de core es muy costosa. Una alternativa puede ser clasificar en los bordes usando el campo Flow Label cuando está disponible para luego procesar de una manera más eficiente.

FbWRED parece solucionar de mejor manera el problema de los tráfico que no responden de forma adecuada a la congestión, como el caso de UDP.

4.6. Ejemplos con disciplinas de encolado

Para ilustrar el uso de las diferentes disciplinas de encolado se agrega a este texto el Apéndice "C".

Capítulo 5

DiffServ (Differentiated Services)

5.1. Introducción

El modelo DiffServ (DS), definido en RFC-2474 [NBBB98], RFC-2475 [BBC+98] y RFC-3260 [Gro02], es un enfoque más nuevo para ofrecer QoS, comparado con el anteriormente visto: IntServ. DiffServ en contraste con el modelo ofrecido por ISA, fue desarrollado para clasificar el tráfico en Internet en clases o grupos más amplios o generales, dando una solución de grano grueso que soporta escalabilidad. Se define un dominio de QoS, una red o un conjunto de estas, dentro del cual el campo DSCP/IP Precedence o Traffic Class tiene un significado específico, permitiendo así que los paquetes dentro del dominio, sean tratados acorde al valor que llevan. En el modelo se especifica un rango de clases DS (DiffServ) y se sugieren los códigos de marcado para las mismas. A diferencia de IntServ, donde se realiza una clasificación por flujo generando una solución con un complejo algoritmo para administrar gran cantidad de hilos simultáneos de datos, este nuevo esquema es más simple.

DiffServ fue desarrollado en los finales de los 90's como respuesta a la necesidad de un modelo de QoS más sencillo y escalable. IntServ ya había mostrado su deficiencias y la complejidad no lo hacia apto para redes de gran tamaño. La QoS con IntServ parecía ser una característica demasiado costosa. Desde el punto de vista de la implementación, el nuevo modelo es mucho más sencillo y, hasta hoy en día, ha demostrado ser fácilmente desarrollado en un gran número de redes. Los mecanismos de DiffServ son los más utilizados en todos los routers dando tratamiento diferenciado tanto a IPv4 como IPv6, incluso se aplican las mismas técnicas sobre MPLS usando

el campo para marcar CoS que inicialmente se consideraba como Experimental.

IPv4 desde su concepción, el RFC-791 o STD-5, consideró un campo en el datagrama para indicar la selección de la QoS. En los principios fue llamado ToS (Type of Service). Luego la codificación e interpretación fue cambiando sobre esta porción de 8bits [1]. Según DiffServ el campo es renombrado a **DSCP (Differentiated Services Code Point)** o para IPv6 como **Traffic Class**. Esta porción del datagrama tiene como objetivo indicar el tratamiento diferenciado de los paquetes para los routers.

En la arquitectura propuesta por DiffServ se utilizan nuevos protocolos y mecanismos para clasificación y marcado de paquetes. En esta la clasificación se realiza en una cantidad reducida de grupos que llevan el nombre de **Forwarding Classes**. Los paquetes deben ser clasificados y marcados al momento que arriban al domino o red de QoS. Un **QoS DiffServ Domain** o **Dominio de QoS DiffServ** es un grupo de dispositivos, probablemente todos formando parte de una misma red, que implementan las mismas políticas de QoS. Cada equipo debe manejar los paquetes perteneciendo a la misma clase de la misma forma, similar como lo hacia IntServ, pero en este caso en grupos mucho más amplios en lugar de hacerlo por flujo. Los nodos, tanto switches, routers como otros dispositivos, involucrados en la QoS bajo una administración común definen el dominio. Una vez que el paquete ingresa a la red, es clasificado y mapeado a una clase de servicio, **CoS**, es marcado mediante campos en el encabezado, en particular el antiguo ToS, renombrado como DSCP, y luego es tratado de acuerdo a este valor. Incluso puede ser remarcado en caso que las condiciones de la red o del tráfico cambien, aunque el remarcado rara vez sucede en el core de la red. Cada clase representará un tratamiento de despacho, en inglés **Forwarding Treatment**, en términos de prioridad de descarte, prioridad de despacho, ancho de banda máximo, ancho de banda reservado y tamaño de las colas para controlar el delay.

El modelo no impone en principio ningún tratamiento ni escala particular de prioridad. Deja estas definiciones al operador de la red, aunque con el objetivo de permitir inter-operabilidad entre dominios recomienda como estándares algunos conjuntos de clases, por ejemplo, “best-effort” que siempre debería estar presente. DiffServ provee un entorno, framework de trabajo, para la clasificación y el marcado en grandes grupo o **Forwarding Classes**.

[1] 8 bits de los cuales algunos se usan para otros propósitos, como ECN

5.2. DiffServ vs. IntServ

DiffServ intenta atacar y solucionar las siguientes deficiencias de IntServ:

Escalabilidad: para IntServ mantener los **soft sates** en cada router para cada flujo es una tarea pesada, más en el núcleo de la red donde la carga de tráfico tiende a ser extrema, al límite de las capacidades de las interfaces. DiffServ opta por un esquema de agrupamiento a mayor escala en grupos más generales sin necesidad de mantener estados.

Modelo de servicio flexible: Intserv solo define dos clases estándares. Diffserv deja abierta las clases y define algunas estándares, las que pueden ser usadas en la implementación directa o como referencia.

Simplicidad/Complejidad: El hecho de que RSVP u otro protocolo de señalización debe ser implementado en cada router en el dominio hace que el protocolo sea más complejo para el caso de IntSetrv. La reserva y liberación de recurso es una tarea que agrega complejidad a la red y atenta en parte con el principio de end-to-end. Para DiffServ, la señalización no es un requerimiento, aunque se puede proveer.

5.2.1. Cómo trabaja DiffServ

Los routers en el modelo DiffServ implementan un mecanismo de tratado de paquetes llamado **Per-Hop-Behavior (PHB)**. Un PHB es una descripción de como externamente se observaría el **Forwarding Behavior** aplicado en un nodo compatible con DS (i.e. un router) dentro del dominio de QoS a un tráfico de una misma clase. PHB define la propiedades del forawding asociadas a un tráfico dado.

Otra definición que encontramos en el modelo es una **Agregación de Comportamiento/tratamiento**, en inglés **Behavior Aggregate (BA)**, término que se refiere al tráfico en sí que pertenece a la misma clase y, por lo tanto, tendrá el mismo tratamiento o PHB. Un BA habitualmente se lo asocia con los paquetes marcados con el mismo código o DSCP (DiffServ Code Point).

Diferentes PHB pueden ser definidas para ofrecer distintos tratamientos a los paquetes de datos. Por ejemplo, baja tasa de pérdida/descarte, bajo delay o latencia,

un ancho de banda asegurado determinado o el simple “best-effort”.

Otro concepto que provee el modelo DiffServ es el **Bandwidth Broker (BB)**. El agente **BB** no es obligatorio en el modelo, pero permite un manejo más sencillo y escalable de las políticas. El **BB** mantiene la información de la política y prioridades de QoS del dominio. Reserva las marcas para cada clase e incluso podría hacerlo con los anchos de banda para las mismas. A través de este se puede derivar la configuración en los nodos de la red para el manejo de la QoS. El mismo también permite lograr QoS multi o inter dominio de forma end-to-end, comunicándose con los BB de los dominios vecinos y “acordando” un mapeo entre los PHB de cada uno. Más adelante ser verán otros detalles de la componente.

Los pasos en los cuales se aplica DiffServ son los que se enumeran a continuación:

1. Primero, la aplicación genera el tráfico. En contraste con IntServ, la aplicación no participa directamente en el proceso para indicar los requerimientos de QoS. La aplicación puede colaborar en el marcado si conoce de antemano el BA indicado por el DS.

2. Una vez que los paquetes llegan a los routers de borde, edge, del dominio de QoS DS, deben ser clasificados y marcados acorde a las políticas de QoS de entrada a la red. Los routers de borde del dominio hacen el trabajo de clasificación en una forma muy similar a como se hace en IntServ: inspeccionan los campos del paquete, incluso del segmento o PDU de la aplicación (cuando es posible), luego harán el “matching” con las políticas y finalmente marcarán el DSCP. La clasificación se hace en las diferentes BAs definidas en la red. El generador puede estar dentro del dominio, por lo que el primer router que trate el mensaje cumplirá las funciones de router de borde.

3. En los routers de borde el tráfico también puede ser acondicionado mediante shaping o policing de acuerdo a las políticas del dominio. Las definiciones de los PHB se pueden hacer de forma coherente pero distribuida en todos los equipos de la red o de forma centralizada a través del BB.

4. Una vez clasificados en un BA, posiblemente acondicionados y marcados los paquetes en su encabezado, seguirán el camino a lo largo del dominio siendo tratados con el PHB asociado al BA al que pertenecen acorde al DSCP. Excepcionalmente podrán ser re-marcados o re-acondicionados.

5. Los paquetes atravesarán el core de la red y serán tratados de igual forma en el resto de los routers del dominio de acuerdo al BA asociado al PHB.

6. Finalmente los paquetes llegarán a destino o saldrán del dominio. Para la salida a otro dominio con QoS, el BB se puede encargar de negociar el mapeo entre las marcas y PHB de la red saliente a la entrante.

5.3. Componentes del modelo DiffServ

En el modelo DiffServ, a diferencia del de IntServ, parece no haber una clara separación entre plano de datos y plano de control, ya que la señalización no se contempla como parte de la definición. Las funciones de la red se separan entre los nodos de borde/edge y los de núcleo/core. Los nodos de borde son los que aceptan el tráfico al entrar o mandan el tráfico al salir del dominio y los de núcleo son los que internamente hacen el proceso de la mayor parte del tráfico. Los de núcleo solo "hablan" con nodos que pertenecen al mismo domino, a diferencia de los de borde que intercambian tráfico con routers de otros dominios. Las operaciones más complejas son llevadas a los bordes donde la cantidad de tráfico se supone menor y se trabaja a menores velocidades. Los routers de core deben tener tareas simples para hacerse lo más rápido posible y en la cantidad que se demande.

5.3.1. Nodos de Borde/Edge

Los nodos o routers de borde actúan al ingresar o sacar tráfico del dominio y tienen asignadas dos tareas principales:

1. Clasificación (Classification)

2. Marcado (Marking)

En la figura 5.1 se muestra un diagrama en bloques de las tareas basado en el diagrama en ASCII Art encontrado en RFC-2475. Además, se muestran dos tareas auxiliares:

1. Medición (Metering)

2. Acondicionamiento (Conditioning): Shaping o Policing

La medición es una tarea necesaria, el acondicionamiento o "conditioning" podría no usarse. La misma se aplica cuando el perfil del tráfico no concuerda con las políticas del dominio.

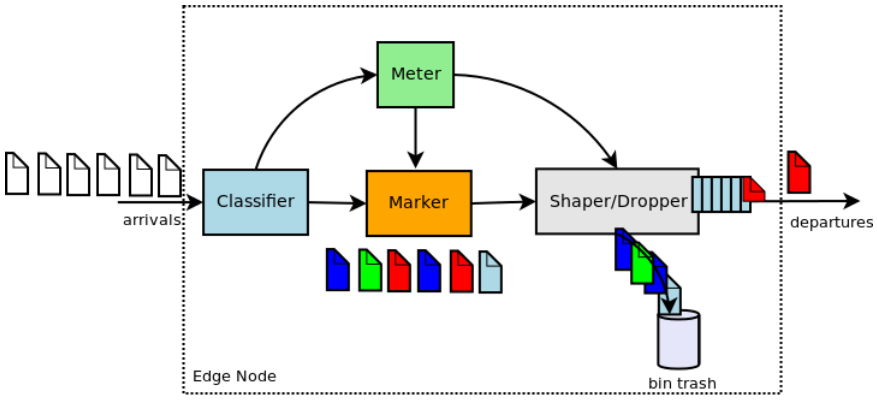


Figura 5.1: Diagrama en bloque de las componentes DiffServ para clasificación y acondicionamiento

5.3.1.1. Medición

La medición, en ingles el "metering", cuenta el tráfico entrante y lo compara con el perfil de tráfico acordado. El módulo pasa información al acondicionador para que este actúe.

5.3.1.2. Marcado/Marking

En el documento RFC-2475 se indica que los paquetes deben ser marcados con un Code Point o codepoint particular de acuerdo al BA. La marca realizada podrá ser relacionado con un único Code Point o tener un conjunto de códigos para el mismo BA. La marca será usada para seleccionar el PHB o el PHB group de acuerdo al estado del medidor. Cuando los paquetes cambian el Code Point se dice que fueron remarcados: "re-marked".

5.3.1.3. Acondicionamiento/Conditioning

En general, el tráfico es acondicionado, policed o shaped, antes de que entre al dominio de QoS para asegurarse que concuerde con las políticas del mismo. El perfil aplicado debe ser derivado de los acuerdos de servicio, **Service Level Agreement (SLA)** (Los detalles de dicho término se verán más adelante en este mismo capítulo).

Como ya se vio en los capítulos anteriores el acondicionamiento puede ofrecerse mediante:

Shapers: retardan los paquetes para hacer cumplir la política. Los encola y los va enviando acorde el despacho.

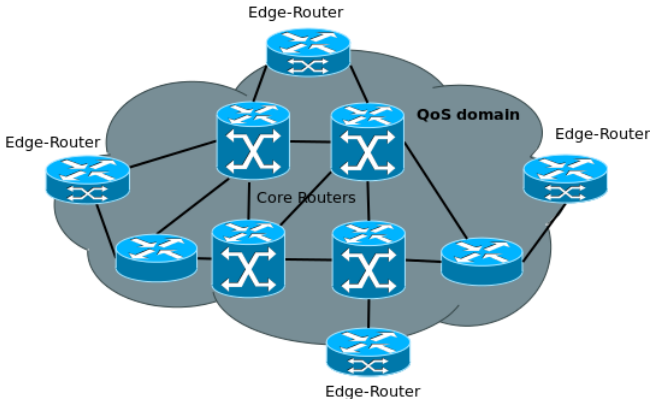


Figura 5.2: Dominio de QoS único

Policers: en lugar de hacer "buffering" de los paquetes, los que no cumplen con la política son descartados, habitualmente se dice "policed".

Otra tarea del acondicionador también puede ser el re-marcado cuando los paquetes están fuera del perfil. En ocasiones el policer en lugar de descartar re-marca, clasificando el tráfico en un BA distinto, probablemente con menos restricciones en el PHB.

5.3.2. Nodos Internos/Core

Tanto los nodos de core como los de los bordes deben ser capaces de aplicar el PHB apropiado a los paquetes de acuerdo a sus marcas. Las marcas se derivarán del SLA. Una vez acondicionado y marcado el tráfico en la entrada de la red, este será transportado a lo largo del domino. Los routers de core manipularan, "forwardarean", los paquetes basados en el PHB configurado según la clase o BA.

Aún en la arquitectura más simple de QoS se asume que la clasificación y el acondicionamiento se realiza en los bordes de la red, aunque estas tareas no son excluidas del núcleo. Por ejemplo, en el RFC-2475 se da el caso de una red con enlaces transoceánicos donde sobre estos se deben aplicar políticas más restrictivas. En la figura 5.2 se modeliza el caso más sencillo de un domino de QoS único.

5.3.3. Bandwidth Broker

Las tareas de reserva y control de los recursos de la red pueden ser realizadas de forma manual, o por agentes que tienen el conocimiento de las políticas y prioridades

de la organización. Las configuraciones manuales son más sencillas, pero no escalan en redes grandes. En el caso en el cual la QoS debe ser expandida a más de un AS (Autonomous System) o dominio de QoS para proveer QoS end-to-end, la configuración manual es aún mas dificultosa. Por las mencionadas razones se incluye un nuevo elemento en el modelo llamado Bandwidth Broker (BB), el cual, en párrafos anteriores ya fue mencionado. La configuración promovida por la comunicación de agentes automáticos es más inteligente y adecuada para casos de grandes redes. Esta nueva entidad es definida en el documento RFC-2638 [NJZ99].

El BB es configurado con las políticas de la organización y se encarga de mantener el registro del cumplimiento de la misma de acuerdo al estado de la red. El monitoreo global es una tarea que también recae en este agente.

Como se mencionó los BB tienen mayor relevancia en la QoS cuando atraviesa más de un dominio. En ese caso se requiere un protocolo de señalización para la comunicación entre los BB "vecinos". Cada organización con un dominio de QoS DS debería implementar un BB con el objetivo de comunicar su política a los dominios adyacentes. Un BB tiene dos responsabilidades:

- Una es distribuir dentro de la red de la organización, donde se aplica la QoS, las políticas en todos los routers y dispositivos participantes. Se indican las marcas, las reservas y el shaping, si se aplica.
- La principal es gestionar la comunicación con los nodos BB de los dominios de QoS DS adyacentes. La asociación entre BBs se denominan regiones de confianza: **Trust Region**.

Como responsabilidad secundaria está el monitoreo del uso de los recurso de la red. Inicialmente, el BB se configura con una base de datos con las políticas a partir de la cual se distribuyen a los nodos de la red. Los BB pueden monitorear y al mismo tiempo cambiar configuraciones de forma dinámica, respetando las políticas de QoS para el tráfico de red de la organización. Básicamente tiene dos tipos de diálogos, uno con los routers y nodos dentro de la organización y otra con los BBs externos.

El BB es una componente extra del modelo y no siempre se encuentra presente. La tarea de los mismos sirve en la configuración y administración de grandes redes, pero la más importante es la de extender la QoS de intra-domain a inter-domain, pudiendo

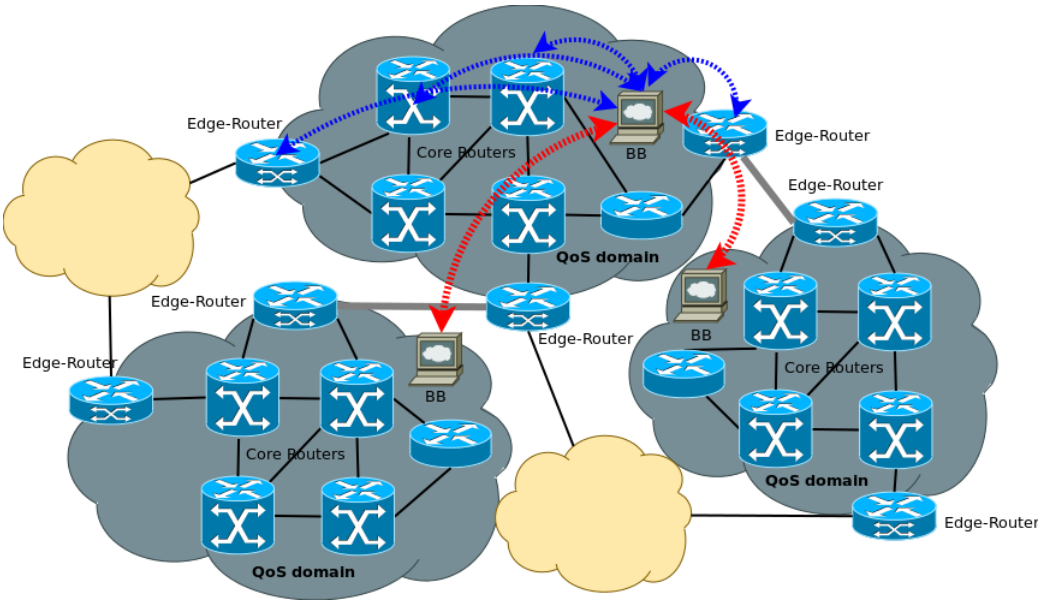


Figura 5.3: multi-dominio de QoS o "trust region"

así ofrecerse un modelo end-to-end. No se tienen registros de implementaciones de BB recientes difundidas en las redes actuales. Se puede mencionar algunos artículos académicos como [KB00] y [SB03] y los experimentos en Internet2 con QBone, `http://qbone.internet2.edu/`, aunque parecen no haber tenido alcance hasta estos días y muestran haber perdido interés. En la figura 5.3 se diagrama el multi-dominio de QoS y los BB.

5.4. Proceso de marcado en DiffServ

En el RFC de IP, RFC-791, se definieron algunos flags en un campo de 8 bits con el objetivo de dar un tratamiento diferente a los datagramas. El campo fue llamado **Type of Service (ToS)**, tipo de servicio, incluido en el encabezado IPv4. La definición solo usaba 6 (seis) bits de los 8, los 3 (tres) primeros para indicar la precedencia: **IP Precedence**, y los restantes como flags para indicar una alternativa entre: Bajo delay, alta confiabilidad, alto ancho de banda digital. Los 2 (dos) restantes fueron etiquetados como "MBZ" (con el sentido "Must Be Zero", (deben ser cero) o "CU" (con el sentido "Currently Unused", sin uso por ahora). Ver la figura 5.4 basada en la documentación del fabricante cisco.

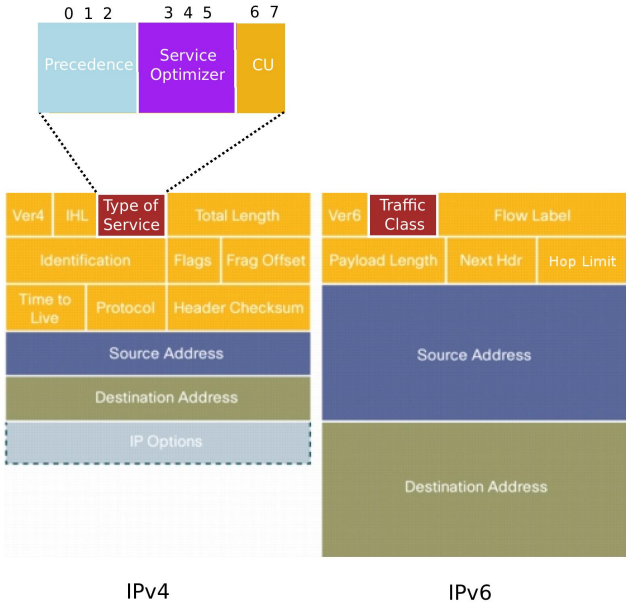


Figura 5.4: Byte Type of Service en el encabezado IPv4 y su contraparte en IPv6

Valor	Nombre
111 (7)	Network Control
110 (6)	Internet Control
101 (5)	CRITIC/ECP
100 (4)	Flash Override
011 (3)	Flash
010 (2)	Immediate
001 (1)	Priority
000 (0)	Routine

Cuadro 5.1: Valores del los bits de IP Precedence

5.4.1. IP Precedence/Precedencia IP

Los bits **IP Precedence** son ubicados como los MSB (bits más significativos): 7, 6 y 5 de acuerdo al almacenamiento en memoria o 0, 1, 2 de acuerdo a IETF). Estos son usados como indicador de prioridad del paquete. La intención es denotar la importancia en el forwarding dentro de la red con respecto a otros datagramas que la cursan. Algunas redes podrían considerar estos valores y tratarlo de manera diferente. La tabla de precedencia se muestra en la tabla 5.1.

La parte del campo precedencia es definida en el RFC-1122 donde se indican los lineamientos para los hosts "Requirements for Internet Hosts – Communication Layers" y su uso en RFC-791. Todos unos (1s) `111(7)` indica **Network Control**

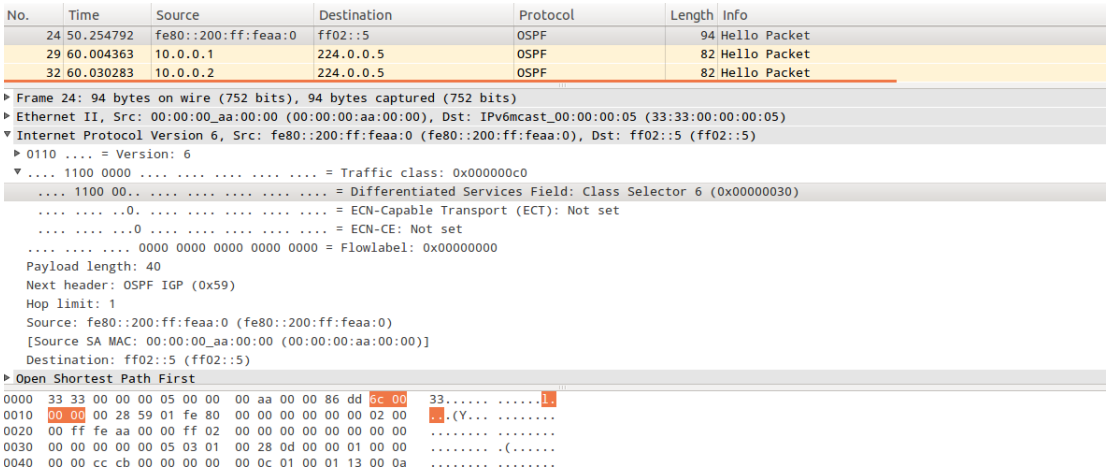


Figura 5.5: Paquete Hello de OSPFv3 con el campo IP Precedence

Precedence (Highest Priority) la prioridad más alta. Es para ser usado por la red y es responsabilidad de la red el uso de este valor.

De acuerdo al RFC-791, **The Internetwork Control Precedence**, indicado por el valor `110(6)`, tiene el objetivo de ser usado por los routers o gateways para marcar protocolos de control. Varios routers de fabricantes actuales lo usan para indicar los protocolos de enrutamiento. La figura 5.5 muestra un paquete OSPFv3 marcado con $0x30 = (110)(000)_2$, donde la precedencia es igual a $0x6 = 110_2$. Recordar que los bits son acomodados en el orden inverso. Para el caso de DSCP la precedencia es interpretada como selector de clase.

Para otros valores de estos bits, la IETF no define su significado en sus documentos. Por su parte el DoD (Department of Defense of The United State of America) define su significado para dentro de sus redes. Lo regla en el documento nombrado como DD173. Las explicaciones que se da del campo Precedence/Priority (Desde la más alta prioridad a la más baja) son las siguientes:

Critic/ECP Precedence: el sentido es para Emergency Call Processing y solo debería ser usado para comunicaciones autorizadas con esa prioridad, por ejemplo, para usar por la United States Government Emergency Telecommunications Service (GETS), la United Kingdom Government Telephone Preference Scheme (GTPS) o otras agencias de gobierno similares.

Flash Override (X): reservado para mensajes relacionados con estallidos hostiles y/p detonadores nucleares.

Flash (Z): reservado para iniciar contactos con el enemigo y operaciones en combate de extrema urgencia.

Immediate (O): reservado para mensajes de situaciones que pueden afectar le seguridad nacional o de fuerzas aliadas.

Priority: (P): para mensajes que requieren una acción expeditiva por parte del destino.

Routine: (R): usado para el resto de los mensajes transmitidos por medios eléctricos.

Todas las descripciones relacionadas con términos militares. Fuera de los 5 (cinco) alcances definidos por el DoD, el sentido es solo una escala de prioridades. Luego la semántica es reemplazada por el Code Point DSCP. Estos tres bits de prioridades son completamente dependientes de la administración de la red de la organización, o en términos de redes del AS(Autonomous System). Sus valores nunca tendrán un tratamiento acorde fuera de la red. Una sencilla regla de como se aplican dentro de un AS es que siempre los de mayor prioridad serán ruteados antes que los de menor. Con solo estos valores da una idea de prioridad, pero no permite por ejemplo, indicar prioridad de descarte, u otro parámetros. Por otro lado en general los valores de Network Control y Control de Precedencia de Internetwork son reservados para paquetes generado por los routers como ICMP. La precedencia no es implementada de forma consistente sobre diferentes redes hoy en día y posiblemente nunca lo será.

5.4.2. Flags de optimización IP/IP Optimize Flags

De forma contigua a los tres bits de Precedencia encontramos otro sub-campo llamado **IP Service Optimize Flags** o **Flags de optimización IP**. Este incluye los bits: 4, 3 y 2 de acuerdo al orden; o 3, 4 y 5 de acuerdo al IETF. Esta parte del byte de ToS fue utilizado para indicar a la red cual de los parámetros, para hacer el ruteo del paquete, se debe considerar más importante. Los parámetros especificados en sus inicios fueron:

- Bajo delay
- Alto ancho de banda digital
- Alta confiabilidad

Bit	Usado para indicar
0	Precedence
1	Precedence
2	Precedence
3	0 = Normal delay, 1 = bajo delay
4	0 = Normal throughput, 1 = alto throughput
5	0 = confiabilidad normal, 1 = alta confiabilidad
6	Reservado
7	Reservado

Cuadro 5.2: Valores del los bits del campo ToS

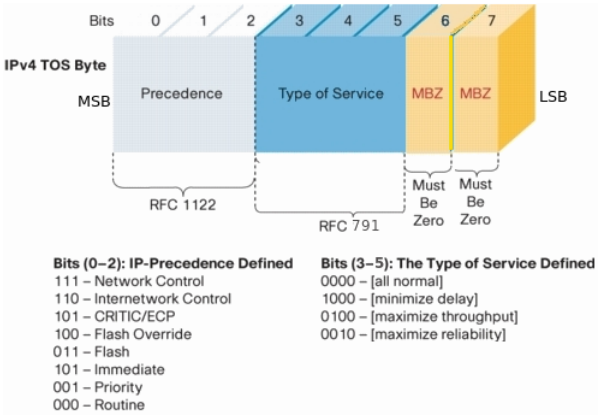


Figura 5.6: Formato del ToS en su definición acorde RFC-1122 y RFC-791

Básicamente, los valores son mutuamente excluyentes, es decir, si uno está seteado (encendido) el resto no lo debería estar, salvo excepciones poco usuales. En el RFC de IP se indican los bits del ToS con sus significados. Ver la tabla 5.2.

Estos nuevos bits a menudo son llamados propiamente como **ToS field**, agregando así algo de confusión con el campo llamado **ToS byte**. También se los encuentra con el término DTR-bits, que proviene de (D)elay, (T)hroughput y (R)eliability. La disposición de los campos se puede observar en la figura 5.6 modificada a partir del JPEG de cisco.com (Notar que en la figura el orden de los bits se muestra de forma inversa, MSB es el 0, de acuerdo a como se presenta en la documentación IETF que adopta el orden en que se transmiten los bits y no como se almacenan en memoria).

5.4.3. Nueva especificación del ToS

En el RFC-1349 [Alm92], luego obsoleta por RFC-2474, el campo ToS recibe una re-definición. La porción de los flags es extendida de 3 a 4 bits, agregando un bit para

Código	Decimal	Significado
100	4	minimizar delay (MD)
010	2	maximizar throughput (MT)
001	1	maximizar confiabilidad (MR)

Cuadro 5.3: Valores de los Flags según RFC-791

Código	Decimal	Significado
1000	8	minimizar delay (MD)
0100	4	maximizar throughput (MT)
0010	2	maximizar confiabilidad (MR)
0001	1	minimizar costo monetario (MMC)
0000	0	servicio normal (BE)

Cuadro 5.4: Valores del los Flags según RFC-1349

el parámetro: **Low monetary cost**, bajo coso monetario. Ver tabla 5.4.

Para ver los detalles consultar la figura 5.7 (Notar que en esta figura también el orden de los bits se muestra de forma inversa, MSB es el 0).

En el RFC-1349 se redefine el ToS como un valor entero en lugar de como un conjunto de flags y la computación del "OR" lógico de dos valores de ToS deja de tener sentido. La semántica de los valores no definidos queda sin especificar. El código 0000_2 es el servicio normal de "best-effort (BE)". Los cuatro flags pasaron a tener el nombre de DTRC-bits , donde la "C" deriva de (C)ost. En la RFC se realizan las recomendaciones para marcar los diferentes protocolos bien conocidos. Se muestran las marcas en la tabla 5.5.

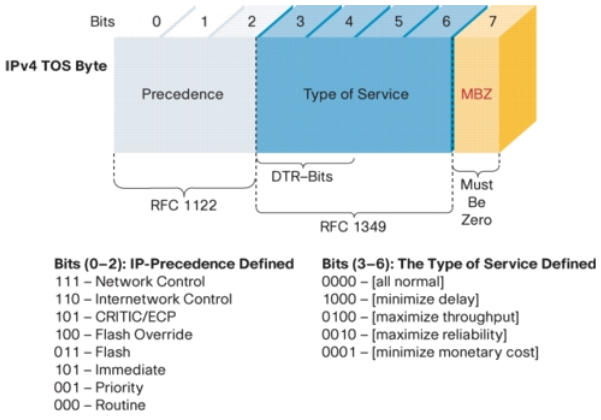


Figura 5.7: Formato del ToS en su definición acorde a RFC-1349

Protocolo	Mensajes	Valor	significado
TELNET		1000	minimize delay
FTP	Control	1000	minimize delay
FTP	Data	0100	maximize throughput
TFTP		1000	minimize delay
SMTP	Command phase	1000	minimize delay
SMTP	DATA phase	0100	maximize throughput
DNS	UDP Query	1000	minimize delay
DNS	TCP Query	0000	best effort
DNS	Zone Transfer	0100	maximize throughput
NNTP		0001	minimize monetary cost
ICMP	Errors	0000	best effort
ICMP	Requests	0000 (mostly)	best effort
ICMP	Responses	= req.	best effort

Cuadro 5.5: Algunos valores recomendados por protocolo para el campo ToS

5.4.4. Especificación de DSCP, re-definición de ToS

En el RFC-2474 se reemplaza significado y el nombre del encabezado ToS. Ahora pasa a llamarse campo DS(DiffServ) y donde los primeros 6 bits (los 6 más significativos) se llaman DSCP (DiffServ Code Point). Se reemplaza lo especificado en RFC-791 y queda obsoleta el documento RFC-1349. El cambio también se aplica a IPv6 y su definición en RFC-2460.

Los 6 bits llamados DSCP son usados como un Code Point para seleccionar el PHB que el paquete debe experimentar al atravesar la red. Los 2 restantes se dejan sin uso y se llaman "CU" (Currently Unused). Estos dos son usados para hacer notificación explicita de congestión por la red, ECN. Si no se usa ECN deben ir configurados a 0 (cero). Ver figura 5.8 (Notar que se sigue manteniendo el orden inverso de acuerdo a como se encuentra en la documentación).

El nuevo formato de DS es incompatible con la definición del ToS de IPv4 acorde a RFC-791 y RFC-1349. Se buscó mantener algo de compatibilidad hacia atrás respetando los 3 bits de precedencia. Esta compatibilidad no está estrictamente asegurada, pero en la definición del RFC se indica. Esto se realizó debido a que el IP Precedence estaba bastante difundido al momento de la propuesta, aunque no necesariamente de forma compatible.

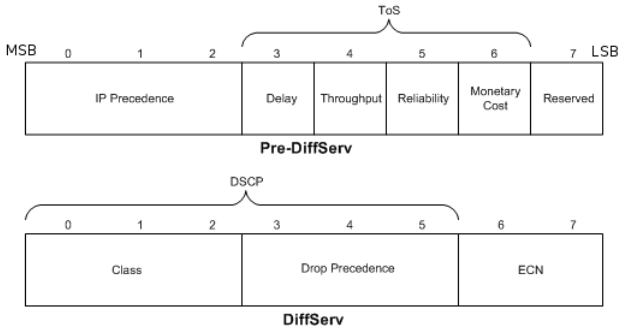


Figura 5.8: Comparación del ToS con el DSCP en su definición acorde RFC-2474

5.4.4.1. Class Selector Code Point

DiffServ define un selector de clase, **Class Selector**, del Code Point, el cual está comprendido por lo 3 bits más significativos (7, 6, 5 de acuerdo al orden en memoria o 0, 1, 2 de acuerdo a la documentación IETF). Los mismos comprendidos por los bits de IP Precedence. Los selectores de clase son de la forma: `XXX000`, donde el símbolo `X` es un bit variable y los demás son 3 ceros. El valor IP precedence puede ser mapeado al selector de clase del DSCP, de esta forma un paquete usando la sintaxis de IP Precedence puede aún ser entendido por un router que trabaja con DiffServ.

Para los selectores de clase de los Code Points los valores numéricos más grandes son los de mayor orden relativo. Esto significa que un paquete con un selector de clase más grande que otro debería tener una mayor probabilidad de ser despachado antes. Esto siempre debería suceder en una condición de carga de la red. Por ejemplo la comparación que se muestra a continuación es verdadera.

```
11X000 > 000000
```

Los códigos DSCP hacen posible mapear múltiples codepoints al mismo PHB. Por ejemplo, todos los paquetes marcados con los 3 bits 0..2 según IETF del selector de clase o precedencia IP tendrían el mismo tratamiento. Esto permite hacer a DiffServ compatible con IP Precedence y así poder coexistir.

Pools	patrón de DSCP	Uso
1	XXXXX0	Recomendación estándar
2	XXXX11	EXP/LU
3	XXXX01	EXP/LU

Cuadro 5.6: Pools de DSCP según RFC-2474

5.4.4.2. Valores IANA DSCP

Los 6 bits del DSCP permiten 64 combinaciones posibles. En el RFC-2474 el IANA considera dividir el espacio plano (flat) en 3 grupos o pools. El IANA usa solo el primero para valores recomendados. El primer pool tiene 32 valores posibles. Los dos restantes se los considera para uso experimental o local, EXP/LU. El tercer pool podría ser usado cuando el primer pool sea agotado. Ver tabla 5.6.

El RFC-2474 asigna 8 (ocho) codepoints recomendados como estándares de la forma `XXX000` que son tomados del primer pool. Estos valores deben al menos acordar los requerimientos de conservar las prioridades para mantener compatibilidad hacia atrás.

Como default PHB todos los nodos deben respetar el procesamiento "best-effort". El codepoint para este es: 000000_2. Los paquetes recibidos con codepoints desconocidos deben ser tratados como si tuviesen este valor.

5.4.5. Proceso de Marcado de DSCP

Como se indicó el proceso de marcado del DSCP ocurre en los bordes del dominio de QoS. Este podría ser aplicado por el nodo origen, aunque rara vez sucede, debido a que el nodo debe conocer los códigos que maneja la red por la que atravesará. Si el nodo origen tiene un SLA con la red en el cual se especifican los DSCP y los PHB asociados, podría directamente marcar desde el inicio el paquete. Los nodos de borde de la red que provee la QoS tienen que tomar dos posibles posturas: confiar en la marca o remarcar de acuerdo a las políticas. Esto también es aplicado cuando el paquete pasa de un dominio de QoS a otro. La descripción de cuan lejos los valores de DSCP son compatibles, se llama **Trust Boundaries** o **Trust Region**.

Para el marcado en los bordes de la red, se debe producir una clasificación previa, la cual se lleva a cabo inspeccionando la información del paquete. En el dominio DS

se tienen dos funciones de mapeo, una que dado un encabezado de paquete, clasificación, da un DSCP y otra que dado un DSCP da un PHB específico. La clasificación puede estar definida como parte del SLA.

Los DSCP estándares permiten a los ISP proveer PHB compatibles mediante manejadores de paquetes bien conocidos, como tipos de colas. Los fabricantes de equipos de red son libres de implementar los mecanismos y parámetros que crean útiles o "marketineros", de acuerdo a sus prioridades. La presencia de los PHB estándares en los routers de diferentes vendedores son soportados con diferentes estrategias de encolados.

5.5. Relación entre DiffServ y SLA

DiffServ hace énfasis en los SLA más que en un proceso de señalización dinámica para indicar la QoS como sucede en IntServ. Los Nodos DS compatibles tratarán de asegurar los parámetros del SLA. Los SLA serán acordados en contratos de long-term.

Previo a aplicar la QoS, el SLA debe ser negociado entre las partes, como podría ser el caso entre cliente y proveedor de servicios de red (ISP o NSP). En el SLA se debe indicar que flujos o que clase de paquetes deberán ser tratados con prioridad y los parámetros para cada clase como ancho de banda, delay o tasa de descarte. Este conjunto de parámetros se llama **Traffic Profile**, perfil de tráfico. El tráfico prioritario entre los flujos es descripto por los valores de los campos en los encabezados a tener en cuenta, por ejemplo, direcciones IP, números de puertos o incluso el DSCP. El SLA finalmente se materializará como parte del contrato. En el RFC-2475 se encuentran las siguientes definiciones.

Service Level Agreement (SLA): un contrato de servicio entre un cliente y un proveedor donde se especifica el tratamiento de los paquetes en el despacho de estos dentro de la red. Entre varias cuestiones cubiertas por el SLA se encuentra que tráfico será tratado con prioridad, el ancho de banda digital, el delay, el descarte y otros parámetros que el tráfico del cliente deberá recibir. El SLA también incluye el acondicionamiento que el tráfico tendrá constituyendo un **Traffic Conditioning Agreement (TCA)**, acuerdo de acondicionamiento de tráfico. El SLA puede ser aplicado ente dominios o dentro de un dominio mismo.

Traffic Conditioning Agreement (TCA): un acuerdo donde se especifica las reglas de clasificación, medición, marcado y descarte o shaping que serán aplicadas al tráfico del cliente. UN TCA tiene en cuenta todas las reglas de conditioning de acuerdo al SLA.

Traffic profile: una descripción de las propiedades temporales que tendrá el tráfico del cliente, como tamaño de ráfagas o velocidad de arribo (rate). Se indican las propiedades del tráfico de acuerdo al clasificador. Sirve para determinar si el tráfico de una clase está dentro o fuera del perfil. El Traffic Profile define también como medirlo para respetar el SLA.

En RFC-2475 se muestra un Traffic Profile de ejemplo basado en la metáfora del token bucket. Este serviría para un servicio de carga controlada, **Controlled-load Service**. Una clase identificada con el codepoint **XXXXXX** usará el token bucket con lo parámetros R y B, donde R es la velocidad, rate y B el tamaño de ráfaga, burst size.

```
DSCP=XXXXXX: Token_Bucket(R, B)
```

En este caso el tráfico fuera del perfil serán los paquetes de la clase que arriban cuando no hay suficientes tokens para ser transmitidos. La misma idea puede ser extendida a modelos más complejos. Por ejemplo, un tercer valor de tokens extras pueden ser usados en determinada circunstancia permitiendo una extensión de los parámetros bases. Similar como sucede con CIR y EIR en Frame-Relay. Para el tráfico fuera del perfil se pueden aplicar diferentes acciones, se puede descartar, se puede shaperar o se puede remarcar y transmitir mapeándolo al tráfico de menor prioridad (una BA considerada inferior).

Según RFC-2475 los perfiles de tráfico son componentes opcionales de los TCA y su uso depende del SLA específico.

En el caso de hacerse SLAs entre diferentes dominios de QoS DS se debe indicar en los mismos los mapeos entre los codepoints, por ejemplo, mediante reglas de remarcado para los perfiles de tráfico dados.

5.5.1. Encolado y planificación de datagramas

La red que ofrezca QoS realizará la clasificación, marcado, shaping y policing del tráfico que pasa por esta. Para el shaping podrá aplicar diferentes técnicas de encolado o para el policing técnicas de descarte.

5.6. Clases Estándares de DiffServ

Actualmente las propuestas de PHB estándares en los documentos RFC de IETF son las que se describen en las secciones a continuación.

5.6.1. Default PHB (Best-Effort)

RFC-2475 define el PHB default como "best-effort". Cada paquete marcado con el valor de DSCP 000000_2 será tratado con el servicio tradicional de IP. En cada red que soporte Diffserv, cualquier paquete que llegue marcado con un DSCP desconocido por el dominio deberá ser remarcado e incluido en el BA default. El PHB debe estar disponible en cualquier dispositivo compatible con DiffServ. Podría llegarse a implementar encolado para este PHB. Si esto se realiza debe existir una cola con al menos recursos mínimos que permitan a los paquetes marcados con 000000_2 ser transmitidos si hay capacidad.

5.6.2. Assured Forwarding (AF) PHB

Assured Forwarding PHB Group, o PHB de despacho asegurado, está definido en RFC-2597 [HBWW99]. En este caso el tráfico de esta clase tiene una baja probabilidad de descarte mientras no exceda el perfil. Puede llegar a exceder los parámetros acordados, pero este exceso será despachado con una prioridad más baja siendo propenso al descarte en condiciones de carga.

Assured Forwarding PHB divide el tráfico en "N'' clases independientes. Actualmente existen 4 (cuatro) definidas. Cada una recibirá un mínimo asegurado de los recursos (buffers, capacidad del enlace, etc). Dentro de cada clase AF los paquetes podrán tener asignado un nivel de descarte. AF define "M'' niveles de descartes o **Drop Precedence** distintos. El nivel de descarte determina la importancia relativa del paquete dentro de la clase. Para el estándar se tienen 3 (tres) niveles de descarte.

En caso de congestión el valor de drop precedence determinará cuales serán más elegibles, en este caso los de mayor valor.

El nivel de despacho, forwarding asegurado de un paquete IP en una clase AF, dependerá de los recursos que tendrá la clase reservados, de la carga de la clase y del nivel de descarte que tenga. Cuando el tráfico llega al dominio DS de QoS, los routers de borde deben controlar la cantidad de tráfico de cada clase AF definida dentro de este. Si se excede se puede remarcar con niveles de descarte mayores. Los niveles de descarte pueden ser re-escritos también dentro de la red. La otra opción es directamente descartar o shapear.

El manejo de las clases de Assured Forwarding debe ser capaz de detectar la congestión. Ante la misma se descarta, encola o remarca. Para manejar el tráfico en exceso en las colas o para el remarcado se pueden usar las técnicas de AQM (Active Queue Management) como RED (Random Early Drop).

A menudo los AF PHB se implementan en un modelo llamado: "QoS de Medallas Olímpicas" en inglés: Olympic Medals QoS Model. Este consta de tres servicios, de mayor a menor: Oro (Gold), Plata (Silver) y Bronce (Bronze). Cuando los paquetes llegan al domino son asignados a una de las clases. El tráfico en las clases de plata o bronce recibirán un servicio menos prioritario. El nivel de descarte se puede implementar con un leaky bucket descripto en el código a continuación:

```
IF (TOKEN_COUNT(BUCKET(CLASS)) > EXCESS_LIMIT(CLASS) ) THEN
        Dgram.DSCP.Drop_Prec := LOW;
ELSIF (TOKEN_COUNT(BUCKET(CLASS)) > 0 ) THEN
        Dgram.DSCP.Drop_Prec := MED;
ELSE -- EMPTY
        Dgram.DSCP.Drop_Prec := HIGH;
END IF;
```

Los codepoint estándares recomendados para el servicio AF son los que se muestran en la tabla 5.7.

	Class 1	Class 2	Class 3	Class 4
Low Drop Prec.	AF11=001010	AF21=010010	AF31=011010	AF41=100010
Medium Drop Prec.	AF12=001100	AF22=010100	AF32=011100	AF42=100100
High Drop Prec.	AF13=001110	AF23=010110	AF33=011110	AF43=100110

Cuadro 5.7: Valores del los DSCP para servicios AF

5.6.2.1. Recomendaciones de Implementación de AF

Cada grupo de PHB AF puede ser implementado dividiendo el total del ancho de banda digital de la interfaz por la que se dará el servicio. Cada grupo o clase deberá tener una cola diferente asignada. Las colas tendrán el ancho de banda reservado, el máximo y la prioridad de descarte definidos. El ancho de banda que no se utiliza por parte de las clases se puede distribuir entre las que si tienen datos. Las colas WFQ podrían ser apropiadas para la implementación, pero lo más adecuado sería mediante una configuración manual mediante CBQ, CBWFQ o HTB. El descarte o re-marcado se puede realizar con WRED.

Los routers de bordes o extremos de la red deben asegurarse de acondicionar el tráfico de acuerdo a los parámetros para cada clase. También deberán realizar el proceso de clasificación y marcado de acuerdo al SLA.

5.6.3. Expedited Forwarding (EF) PHB

Expedited Forwarding (EF) PHB, PHB de despacho acelerado, es otra componente fundamental de la arquitectura DiffServ. Este PHB está pensado para tratar tráfico con requerimientos de bajo delay, bajo jitter y baja pérdida. Se trata de asegurar que el tráfico que pertenece a la clase sea servido con la mayor prioridad. Este PHB fue definido primero en RFC-2598 [JNP99], luego obsoleta por RFC-3246 [DCB+02]. Este PHB es similar al modelo de IntServ **Guaranteed Bandwidth Service**. A diferencia de AF solo se define una clase para este PHB. El servicio puede usarse para simular conexiones punto a punto sobre una "virtual leased line", como es el caso de los servicios de VoIP u otros servicios de tiempo real.

El PHB especifica que la velocidad de arribo del tráfico de la clase debe ser igual o menor que cierto parámetro $''R''$, independiente de la intensidad del tráfico de las otras clases. Deben ser servidos a la velocidad dada o superior. El tráfico que exceda

el valor deberá ser descartado o tratado fuera del profile como menos prioritario.

Solo las aplicaciones críticas deben ser clasificadas en esta PHB. El valor recomendado para el DSCP del tráfico EF es `101110` que corresponde a `46`. El PHB EF es descripto como el servicio Premium y en el modelo de medallas olímpicas, se lo coloca sobre el Oro, referido como Platino (Platinum).

5.6.3.1. Recomendaciones de Implementación de EF

Un simple enfoque para implementar servicios de tipo EF es mediante un esquema PQ (Priority Queue). El tráfico EF debe ir a la cola de mayor prioridad. La cantidad de tokens para la cola debe ser limitado con el objetivo de evitar la inanición de las demás. El pico máximo de tráfico para la cola debe ser limitado. Una solución con una cola LLQ (Low Latency Queueing) sería una buena implementación.

Siempre debe existir un mecanismo para limitar el tráfico en exceso, ya sea remarcándolo o descartándolo. El shaping no es adecuado para el tráfico con demanda de baja latencia. Debe ser estrictamente controlado el tráfico que ingresa a la red. Una regla general puede ser solo permitir el 30 % del ancho de banda total para la clase, aunque esto dependerá del ancho de banda total y los servicios ofrecidos por la interfaz. El manejo del tráfico EF debe ser mucho más estricto que el AF en cuanto al exceso.

5.7. Mapeando DiffServ a Capa de Enlace

En la arquitectura DiffServ los nodos finalmente terminarán empleando equipos de capa 2 (L2: nivel de enlace) para pasar el tráfico dentro de la red. Es recomendable extender el marcado que se realiza en capa 3 (L3: nivel de red, IP) a capa 2. De esta manera se tiene una implementación más exacta de la QoS. Esto se puede realizar siempre que los protocolos de enlace soporten QoS.

Algunas alternativas, hoy ya en desuso, son ATM y Frame-Relay. Las más usadas son Ethernet con 802.1p, 802.1Q, Wifi con 802.11e y también MPLS con el campo conocido anteriormente como Experimental, RFC-5462 [AR09]. A menudo varios fabricantes agregan mecanismos que permiten mapear valores de L2 a L3 y viceversa. Los campos de marcado de QoS a nivel de enlace se conocen como CoS (Class of

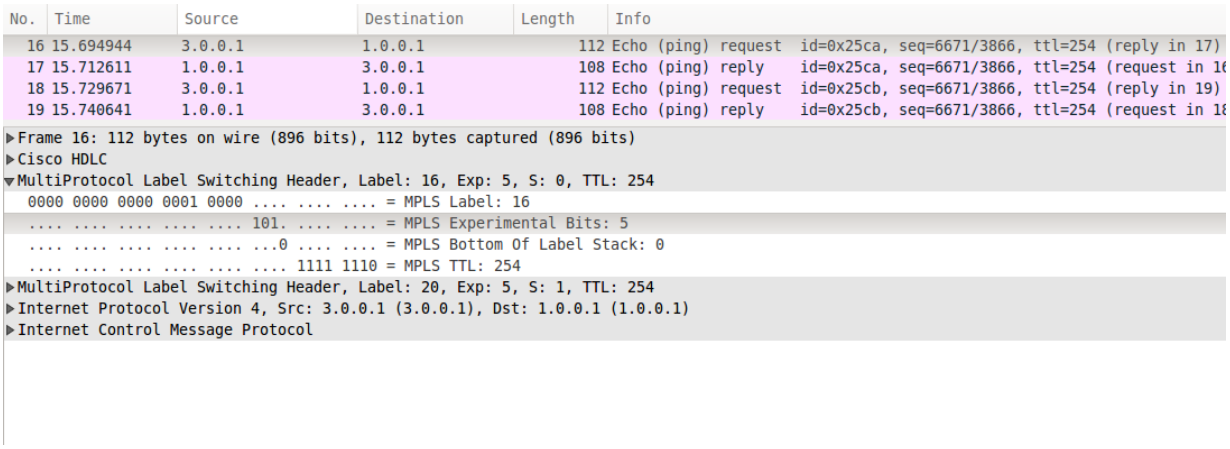


Figura 5.9: Tag MPLS con el campo de ToS/Exp

Service). En la figura 5.9 se muestra un ejemplo de captura de un ICMP "encapsulado" en MPLS y en la figura 5.10 se muestra una trama Ethernet con la marca de prioridad en el tag 802.1Q.

5.8. Ejemplo con DiffServ

Para terminar el análisis de DiffServ tratados en este capítulo se puede consultar en el apéndice "D" ejemplos de como verlo en acción.

5.9. Conclusiones sobre DiffServ

DiffServ es una arquitectura de QoS más moderna que intenta resolver las falencias de IntServ. Resuelve fundamentalmente el problema de la escalabilidad. Hoy es un modelo ampliamente difundido aunque para QoS end-to-end su implementación se torna compleja. DiffServ termina siendo un modelo más simple que IntServ. En este, el tráfico entra en la red (dominio de QoS) y es clasificado en los bordes, agrupándose diferentes flujos en clases más amplias, BA (Behavior Aggregate) que luego serán tratadas por PHB (Per-hop-behaviors). Tiene como características principales:

- Granularidad gruesa.
- Trabaja básicamente de forma estática, sin señalización.
- Podría utiliza QoS/Bandwidth Brokers para administrar y negociar los requerimientos, comunicar con los routers de borde (edge routers). y "tracear" las reservas de recursos o podría implementarse de forma estática.

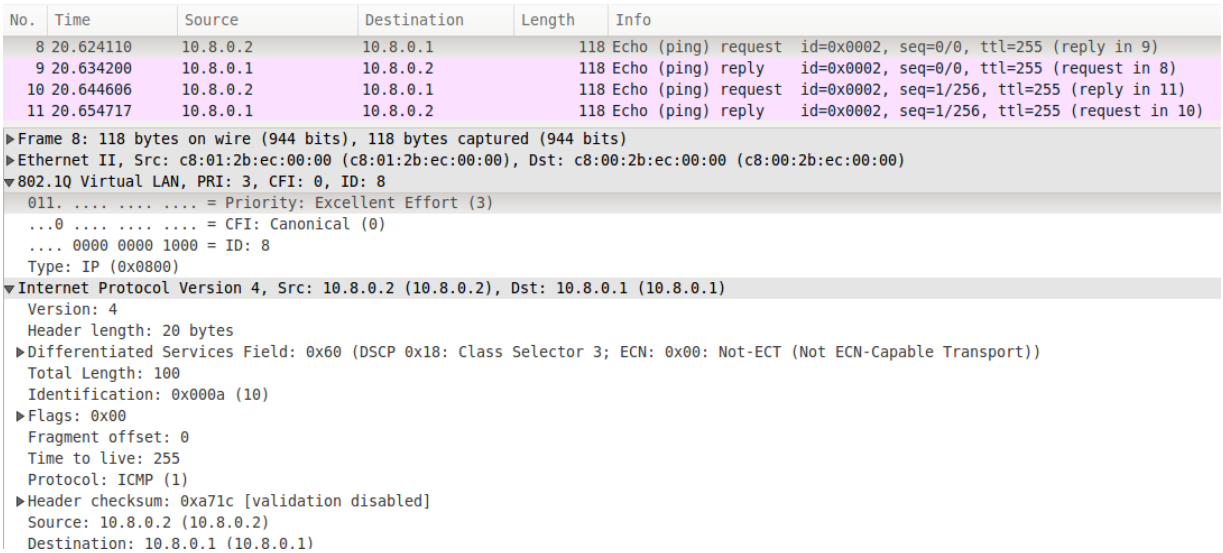


Figura 5.10: Tag 802.1Q con marca de prioridad/CoS 802.1p

- El tráfico, una vez marcado, es tratado dentro del dominio de acuerdo a su marca. Todos los paquetes de la misma clase deberían ser tratados de la misma forma dentro del dominio de QoS.
- DiffServ define algunos tratamientos estándares como "best-effort", AF y EF.
- Básicamente se utiliza el campo DSCP (Differentiated Code Point) en IPv4 o Traffic Class en IPv6.
- Es un modelo más sencillo y más escalable que IntServ.

Sus desventajas son:

- Se agrupan flujos individuales en una misma clase, no pueden ser diferenciados. Pocas clases.
- Modelo más estático en la implementación.
- Puede producirse un delay mayor en el mapeo a las clases.
- Difícil para adecuar correctamente a un modelo de end-to-end (extremo a extremo).

Capítulo 6

Análisis de propuestas de utilización del Flow Label

En este capítulo se analizan varios documentos que regulan o proponen algunas formas de uso del campo Flow Label en IPv6. Se intenta hacer una revisión de varios artículos analizándolos en orden cronológico. Se comienza por el estándar, descartando documentos previos, aunque se hace mención de aquellos que resultan relevantes. Los documentos analizados más profundamente son los que se enfocan en el uso del campo para QoS. Para aquellos que proponen el uso para otros fines solo se los mencionan y/o se realiza una breve descripción. Se redacta una sección del capítulo por cada documento analizado.

6.1. Protocolo de Internet, versión 6 (IPv6)

En la definición del documento draft estándar para el protocolo IPv6, "Internet Protocol, Version 6 (IPv6) Specification" RFC-2460 [HD98] de 1998, se hace mención del campo Flow Label en la sección 6 (seis) del mismo. En el párrafo se indica que debe ser usado por el origen para requerir que determinada secuencia de paquetes tengan un tratado diferente por los routers IPv6, como solicitar una QoS diferente a la normal, que es "best-effort", o incluso requerir un servicio de tiempo-real más estricto. Se señala que su uso es experimental y que los equipos que no sepan como tratarlo deberán ignorarlo en el caso de ser "middleboxes", o desde el origen dejarlo configurado en 0 (cero). Queda registrado en el RFC que el campo tuvo alguna intención de uso en el marcado de QoS.

La Semántica para el campo indicada en el documento está definida en el Apéndice "A" del RFC:

- Define un flow como una secuencia de paquetes donde el origen requiere el mismo tratamiento especial por los router intermedios. Indica que este tratamiento podrá ser convenido por un protocolo de control.
- Un flow es identificado por las direcciones origen, destino y el campo Flow Label distinto de cero. Los paquetes que no son identificados con un flujo deben llevar el valor 0 (cero) en el campo.
- El valor debe ser asignado por el origen, debe ser generado de forma pseudo-(aleatoria) entre los valores 1 y 0xFFFFF. Indica que debe ser útil para funciones de hashing en los routers.
- Paquetes pertenecientes al mismo flujo deben ser tratados de la misma forma. Todos deben compartir las mismas opciones/cabeceras de extensión, salvo por cabeceras de extensión hop-by-hop y next-header de Routing Header.
- Se puede chequear por los equipos intermedios que las opciones llevadas estén de acuerdo a las requeridas por el RFC estándar. Sino se cumple, un mensaje ICMP podría advertir del problema.
- El tiempo de vida estará definido como parte del establecimiento de un estado del flujo: Flow State, que estará manejado de forma separada al plano de forwarding.
- Se indican precauciones contra el re-uso de valores del campo ante arranques y re-arranques del sistema.

El RFC-2460 fue actualizado a partir de RFC-3697 [RCCD04] donde se define de forma específica el uso del campo. Actualmente el documento RFC-6437 [AJCR11] lo redefine aunque sin grandes cambios. Es importante mencionar que aún el protocolo IPv6 sigue siendo draft estándar y no ha pasado al estado de estándar, si bien es Standards Track ya hace mucho tiempo.

6.2. Nuevas posibilidades ofrecidas por IPv6

Ya hace un tiempo, como en el artículo "New Possibilities Offered by IPv6" [LS98], se cita la capacidad de usar el Flow Label para dar un tratamiento especial al tráfico, como puede ser una QoS diferente a la default. Indica que se puede realizar por medio de protocolos de reserva de recursos o hop-by-hop, como podría hacerse con

DiffServ [BBC+98]. El Flow Label permite diferenciar distintas conexiones entre los mismos extremos sin necesidad de inspeccionar campos de los protocolos superiores, comúnmente, los puertos en transporte. En el documento se indica que los paquetes con el mismo Flow ID, dado por la 3-tupla: direcciones origen, destino y Flow Label; deben tener las siguientes propiedades:

- Son asociados con el mismo destino, por lo que se transmitirán vía el mismo próximo salto (next-hop).
- Pertenecen al mismo grupo de reserva o clase de tráfico (queuing class).
- Deberían tener las mismas opciones y encabezados de extensión de routing (si estuviesen presentes).

Se menciona como optimización para la performance que los paquetes pueden ser transmitidos en base a la información de cache en los equipos intermedios, y si estas se conforman a partir de la re-definición del Flow ID usando el nuevo campo se podría ahorrar un tiempo importante en el proceso de switching y en la memoria requerida por las tablas temporales usadas para el routing/forwarding en capa 3.

El documento solo analiza de forma superficial las posibilidades que ofrece IPv6, es solo un bosquejo y no detalla en ninguna parte del uso de los agregados. Mucha de la información que tiene ya no es útil, pero sienta un precedente en el uso de la QoS en IPv6 con el campo Flow Label.

6.3. RSVP y Servicios Integrados (ISA) con el Flow Label de IPv6

El Internet draft "RSVP and Integrated Services with IPv6 Flow Labels" (draft-berson-rsvp-ipv6-fl-00) [Ber99] hace una propuesta de agregar a RSVP la posibilidad de trabajar con IPv6 utilizando el Flow Label. Define un nuevo `FILTER_SPEC` para ser usado en la reserva de los recurso y marca la necesidad de tener una API bien definida para trabajar con el Flow Label en IPv6. Menciona las posibilidades de generar los valores como (pseudo)-aleatorios o tratando de que sean únicos por origen. Para las modificaciones en el API propone que el sistema operativo maneje los valores y obtenerlos mediante una system-call, otra es mediante la llamada `setsockopt()` para requerir la opción en el socket. También sugiere que el kernel los asigne automáticamente cuando encuentra un comportamiento de flujo. No queda claro como el kernel

descubre este comportamiento, pero lo que se busca es que este trate de no repetir los valores, de la misma forma como lo hace con los números de puertos para la capa de transporte. Como tercer agregado del draft es incluir nuevos mensajes en RSVP.

El documento queda como draft y en la actualidad RSVP para IPv6, y en general RSVP con IntServ, no se ve que haya tenido mucho desarrollo. A modo de ejemplo de la falta de desarrollo, cisco menciona que RSVP en su equipamiento no soporta IPv6. Si bien puede ser transportado sobre el mismo no tiene reserva de recursos para el protocolo de red. Probablemente RSVP-TE para IPv6 tendrá mayor auge. Por ahora implementaciones que utilicen el Flow Label, no parecen verse.

6.4. Una propuesta para la Especificación del Flow Label de IPv6

En la propuesta de RFC, Internet draft: "A Proposal for the IPv6 Flow Label Specification" (draft-conta-ipv6-flow-label-02) [CR01b], de Conta y Carpenter se plantea la posibilidad que el campo Flow Label tenga un carácter bivalente de acuerdo a su uso:

- Extremo-a-Extremo (End-to-End).
- Salto-a-Salto (Hop-by-Hop).

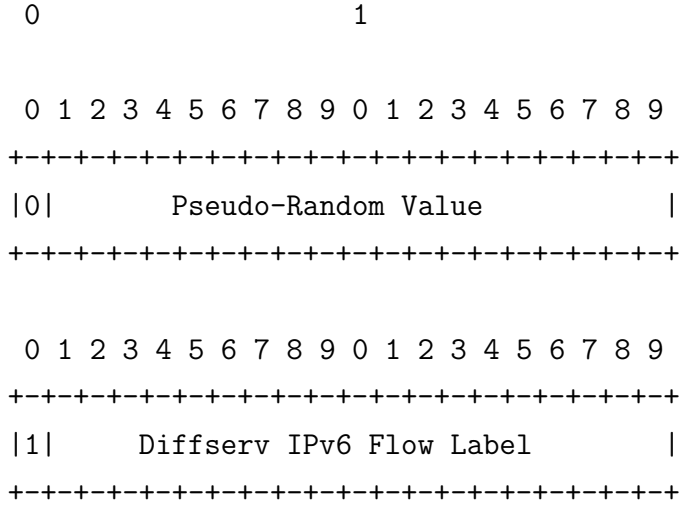
Indica que debe existir la posibilidad que el mismo se puede cambiar durante el transcurso de la transmisión en la red, lo cual requiere que varias de las disposiciones definidas por el anterior estándar IETF, RFC-3697, deban ser suprimidas. Algunas modificaciones se encuentran en RFC-6437 aunque no alcanza la flexibilidad requerida por el documento analizado en esta sección. Algunas de las características requeridas son:

- Un Flow es identificado por la combinación de dirección origen, destino y un Flow Label distinto de cero.
- Un Flow Label cero no tiene significado alguno. No se usa el Flow Label.
- El valor del campo es asignado por el nodo origen, pero puede ser cambiado durante su ruteo por los equipos intermedios, con la condición que el significado original se conserve. Si no se tiene un mapeo el valor no debe ser cambiado.

Como condición se debe lograr que el destino reciba el valor esperado, que será el indicado al inicio del trayecto del mensaje.

- El valor del mismo debe ser entre 1 y 0xFFFFF. Se debe permitir que el valor se comunique a los equipos en el camino, como podría ser vía un mecanismo de señalización, el caso de un protocolo de reserva de recursos, como RSVP o de distribución de etiquetas, LDP. La otra alternativa es que pueda ser configurado en los equipos del camino hasta el destino o transmitido de forma que lo nodos intermedios tengan noción de su significado y tratamiento probablemente de acuerdo a un SLA.
- Paquetes de un mismo flujo deberían ser marcados de la misma forma, aunque pueden surgir agregaciones.
- Se puede chequear por los equipos intermedios que las opciones llevadas estén de acuerdo a las requeridas por la RFC estándar. Sino se cumple un mensaje ICMP podría advertir del problema.

Se menciona como uso del campo en un modelo DiffServ en IPv6, pero a diferencia de marcar sobre el campo Traffic Class, se lo hace sobre el Flow Label. Para indicar cual de los caracteres/significado tiene el campo, si de Extremo a Extremo o DiffServ, se propone marcar en el bit de encabezado (LSb) un 0 (cero) para cuando la información que lleva respeta lo definido por el estándar actual de IETF, donde los restantes 19 bits posiblemente lleven un valor (pseudo)-aleatorio que no debería cambiarse; y un 1 (uno), para el caso que lleve información para la clasificación en el modelo DiffServ.



El valor del campo de DiffServ para el modelo en IPv6 debe ser derivado de acuerdo a lo definido en RFC-2474 [NBBB98] o asignado por el IANA. Debido a la longitud un tanto arbitraria, 19 bits, en el documento se termina reduciendo el código a 16 bits dejando los restantes reservados.

De acuerdo al documento, la primera etapa es cuando el paquete sale desde el origen, en este caso señala que la aplicación puede marcarlo o directamente el stack TCP/IP debería hacerlo. Si esto no se cumple el primer router en el camino debería hacerlo. Cuando el paquete entra en una red con modelo de QoS de acuerdo al Flow Label esta debería clasificarlo y tratarlo de acuerdo al valor y a las direcciones. Podría sufrir un re-mapping al cambiar a otra red.

En el documento se hace hincapié en el uso de una clasificación MF (Multi-Field classifier) a diferencia del BA (Behavior Aggregate) Classifier que solo lo hace en base al DSCP Code-Point para el caso DiffServ en IPv4. En este caso el MF dentro del dominio de QoS puede trabajar directamente sobre la 3-tupla: direcciones origen, destino y Flow Label; indicado habitualmente como Flow-ID en IPv6. Desde un dominio a otro probablemente tenga que examinar más campos y entrometerse en valores de capas superiores si tiene acceso.

El manejo presentado termina siendo básicamente el mismo que el modelo DiffServ, salvo que indica que a diferencia del campo Traffic Class con el valor del DSCP, se puede marcar en el Flow Label información que no es mapeada de forma local solamente. Por lo tanto se puede ofrecer una QoS pseudo extremo-a-extremo. Pseudo es una apreciación del autor de la tesis, debido que si el paquete atraviesa diferentes dominios de QoS probablemente será modificado o tratado de manera diferente. Esto es cubierto por una de las reglas para regir la modificación enunciada anteriormente.

El valor del campo cuando no es configurado por el origen probablemente deberá ser configurado de acuerdo a la 5-tupla que define un flow:(direcciones, puertos y protocolo; o mediante un mapeo local o derivado de un PB (Policy Broker) posiblemente a partir de las reglas de un SLA o SLS. Eso no queda bien definido.

El documento analizado aquí fue presentado como un draft en el 2000, tuvo algunas modificaciones hasta 2001 pero finalmente nunca logró el estado de draft estándar y solo quedo como draft terminado. Deja varias cuestiones sin definir y no se encuentran

implementaciones de referencia del modelo. Es importante mencionar que como autor del documento analizado figura B. Carpenter, autor también del antiguo estándar RFC-3697 y del actual RFC-6437 que intentan estandarizar el uso del campo.

6.5. Un modelo para el uso en DiffServ de la etiqueta de especificación del FL IPv6

En el draft "A model for Diffserv use of the IPv6 Flow Label Specification" (draft-conta-diffserv-ipv6-fl-classifier-01) [CR01a], se plantea una forma de usar el campo Flow Label y el Traffic Class/DSCP para indicar un tratamiento para el tráfico de acuerdo a contratos pre-establecidos. Se especifica un modelo conceptual para el uso del campo incluido en IPv6 con servicios diferenciados (DiffServ). Se muestra un posible método de clasificación basado en el Flow ID dado por la 3-tupla: dir. origen,dir. destino, Flow Label y define una serie de reglas de como se podría utilizar el nuevo campo. Brinda algunos ejemplo ilustrativos.

En las secciones iniciales indica que los valores para el Flow Label o los rangos de valores, deben ser conocidos por las dos partes:

- El cliente/usuario de red, quien requiere y contrata un servicio de QoS.
- El proveedor de servicios de red, quién implementa los mecanismos para satisfacer la demanda/necesidades que el cliente contrata.

Los valores deben ser marcados en los paquetes por el cliente/usuario, llamados "host flow label values". Del lado de la red los valores son usados para configurar los clasificadores en los routers y equipos que transportarán los datos, llamados "router flow label values". La distinción parece estar realizada solo del lado de donde se los usa, pero los valores deberían coincidir. Estos valores deben ser cubiertos por acuerdos contractuales entre usuarios y proveedores de servicios de red, básicamente por: SLAs, SLSs o TCAs.

El modelo presentado lo divide en 2 partes, una llamada: "Host Conceptual Model for the Diffserv Flow Label" y la otra "Router Conceptual Model for the Diffserv Flow Label". Cada modelo se encargará de tareas diferentes, pero funcionarán en conjunto definiendo el modelo completo.

Para el modelo de Host indica que estos son los encargados de establecer el valor en los paquetes que envían y lo pueden asignar de acuerdo a estos tres criterios:

- Valor arbitrario.
- Valor (pseudo)-aleatorio.
- Valor asignado por el IANA.

Los valores arbitrarios posiblemente estén relacionados con un acuerdo, por ejemplo, un SLA. Lo mismo se indica para los (pseudo)-aleatorios y los asignados por el IANA. Si se implementa el modelo DiffServ deberán estar estrictamente configurados de acuerdo a los contratos entre las partes: SLAs, SLS o TCAs. El valor puede ser colocado en el paquete por la aplicación o por el stack TCP/IP del host origen. Como es de esperar los paquetes del mismo flujo deberán tener el mismo valor. Si comparten los paquetes las mismas direcciones, puertos y protocolo de transporte deberán tener asignado el mismo Flow ID. De acuerdo al API para que la aplicación pueda cumplir esta tarea se indica que será la de sockets usando el campo `sin6_flowinfo` de la estructura `sockaddr_in6`. Posiblemente para protocolos orientados a conexión mediante las llamadas `bind()` o `connect()` y puede ser obtenido mediante `accept()`. Para el caso de "connectionless" a través de `sendto()`, `write()` o similares y será obtenido mediante `recvfrom()` o `read()`. También como alternativa plantea otras llamadas como `getsockname()`.

Para el modelo de Router el mecanismo a aplicar será el de configurar el clasificador basado en el Flow Label (las reglas de clasificación) y como procesarlas. Define como clasificador de Flow la siguiente estructura:

```
C:  (Source Address     , Source Address Prefix,
     Destination Address, Destination Address Prefix Length,
     Flow-Label)
```

El clasificador en el router puede configurarse de forma manual o mediante algún mecanismo dinámico, como a través de un NMS (Network Manager System), e.g. SNMP. La regla para el clasificador conceptualmente lo define como:

```
Incoming packet header: (Source Address,
                         Destination Address,
                         Flow Label)
```

Luego la regla será la encargada de tomar la decisión de como tratar al paquete perteneciente a un flujo determinado.

Se presentan algunos escenarios de uso. Un caso es donde una red de acceso debe brindar QoS al tráfico de los clientes que salen de la red. En este caso los routers de acceso (edge) de la red, a los cuales le llega primero el tráfico de los extremos deberán hacer una clasificación del mismo. Si los clientes marcan el DSCP/Traffic Class, el edge puede clasificar en base a este, sino deberá aplicar una clasificación MF (multi-field classification). En el ejemplo si el Flow Label está configurado el proceso podría basarse en su valor de acuerdo al SLA. Otro escenario es donde varias redes de acceso deben pasar por una red de carrier. El ejemplo es similar, pero ahora los routers de ingreso de la red de carrier como los routers de borde de la red de acceso deben tratar el tráfico marcado. Los valores del Flow Label usados deberían reflejar los contratos entre los clientes y las redes de acceso. Podría existir una re-escritura por los routers de borde de las redes de acceso pero debería conservarse el significado original que tuvo. Los "host flow label values" a través de los "router flow label values" entregados a los routers de ingreso a la red carrier deben respetarse.

Finalmente el documento comenta una posible integración con el modelo IntServ. El Internet draft tuvo una modificación pero nunca alcanzó el estado de estándar track. La primera versión del draft, 00, fue titulada "A definition of a IPv6 Flow Label classifier Specification". El autor, A. Conta, es autor también del draft citado [RC01] y de varios documentos que han alcanzado el estado de RFC, como son: RFC-2473 "Generic Packet Tunneling in IPv6 Specification", RFC-3032 "MPLS Label Stack Encoding" y RFC-4443 "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification" [CDG06]. El principal documento del autor relacionado con el tema tratado es el RFC-3697.

6.6. Una propuesta para la Especificación del Flow Label de IPv6

El Internet draft de J. Rajahalme y A. Conta:"An IPv6 Flow Label Specification Proposal" (draft-rajahalme-ipv6-flow-label-00) [RC01] indica que el campo IPv6 Flow Label fue diseñado para permitir una clasificación eficiente de paquetes, y, en el caso de no usarse la misma se debe llevar a cabo inspeccionado los encabezados de capas superiores. De esta forma puede suceder que los campos no estén disponibles debido

a una tunelización/protección, sin descartar la violación entre capas de protocolos, aunque esta última no es algo que realmente preocupe. El documento presenta varias características que debería tener el Flow Label que luego son llevadas al primer Internet Draft. Presenta la idea de un sistema para manejar los Flows a partir de Flow States que se deben crear en los sistemas intermedios a partir de protocolos de señalización como RSVP o SIP/SDP, aunque no especifica nada en particular sobre estos. El manejo particular para cada flujo debe ser aprendido por algún protocolo de control ejecutando en los routers o directamente por información en alguna opción hop-by-hop llevada por los paquetes. Las reglas para el filtrado/clasificación y el manejo de los flujos ("special handling") serán parte del mismo Flow State, como los parámetros bandwidth, delay, etc. El Flow State en el plano de datos/forwarding estará compuesto al menos de:

3-tupla (Info de selección, Flow ID): Source Address, Flow Label, Destination Address

Flow Accounting Info: contador del número de bytes o paquetes para el flujo.

Forwarding Treatment: define el "special handling" al cual los paquetes del flujo estarán sometidos.

El documento indica, igual que la RFC estándar, que los routers no deben asumir ninguna propiedad sobre los valores asignados por los hosts y que no deben afectar su accionar sobre como se distribuyen las etiquetas. No asigna un tiempo fijo a la existencia del Flow State, pero indica que pueden expirar o borrarse en el caso de que no se usen. En el estándar se usa el valor 120 segundos como un tiempo para un flujo. Se menciona que no es necesario definir ningún formato interno para el Flow Label y que cada implementación podría tener el suyo. También indica que no requiere valores (pseudo)-aleatorios, algo que el estándar parece recomendar.

Menciona que los paquetes de un flujo pueden ser mapeados a un Behaviour Aggregate (BA) permitiendo que los router directamente identifiquen por el DSCP de acuerdo a DiffServ y pasen por alto la clasificación por Flow ID.

El documento quedo solo como draft, aunque parece que algunas características se llevaron al estándar. El mecanismo de Flow States parece hoy ya dejado de lado de las especificaciones de la IETF, aunque no se prohíbe. Los autores son finalmente también los mismos del primer Standard Track sobre el campo Flow Label en IPv6, RFC-3697.

6.7. Una especificación modificada para el uso del Flow Label en IPv6 con el fin de proveer una eficiente QoS usando una propuesta híbrida

En "A Modified Specification for use of the IPv6 Flow Label for providing An efficient Quality of Service using a hybrid approach" (draft-banerjee-flowlabel-ipv6-qos-03) [BSM02] se sugiere una especificación pragmática para el uso del campo Flow Label en un esquema híbrido incluyendo varias opciones para usarlo como IntServ, así también como DiffServ o de extremo a extremo en el aprovisionamiento de QoS. El autor define varios formatos de etiquetas, todas con un encabezado fijo de 3 bits, por lo cual quedan 17 bits del campo para marcar. Esto da lugar a diferentes propuestas y, de acuerdo a los encabezados, será la codificación y el significado de los 17 restantes.

Define un encabezado para dejar el flow label en 0 (cero) en el caso que no se marque. Sería la opción default. Agrega otro encabezado para manejar las etiquetas de forma (pseudo)-aleatoria de extremo a extremo, similar como lo indica el estándar IETF.

Otra posibilidad es indicar en el encabezado de 3 bits que el valor al que se debe prestar atención es el del encabezado salto a salto (hop-by-hop), header de extensión de IPv6, como se sugiere en [BS02a] para proveer la QoS. Los 17 bits restantes deben ser ignorados con este encabezado. En este caso el modelo de QoS sería el de IntServ, donde se ofrecen los servicios:

- Guaranteed flow service (Servicio Garantizado).
- Controlled Load Service (Carga Controlada).

Como alternativa se encuentra otra que indica utilizarlo como PHB-ID (Identificador Per-Hop-Behaviour) similar al modelo de DiffServ para QoS. Para esta opción solo se usan 16 bits de los 17. La alternativa es como la indicada en el draft citado en [RFB01].

Aporta también como opción interesante para marcar requerimientos de recursos en donde,uno de los encabezados posibles indica determinados parámetros para la QoS. Por supuesto en el Label se puede solicitar:

- Ancho de Banda Digital, Bandwidth: (expresada en múltiplos de kbps).

- Requerimientos de Buffering, Buffer requirements: (expresado en bytes)
- Retardo máximo, Delay: (expresado en nanosegundos).

Algunos valores podrían relajarse, con un bit inicial que indica si se requieren estrictamente los atributos, sea el caso si la aplicación es de RT blando (soft) o estricto (hard).

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0 1|R|      Flow Label                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Donde $HD = 101$ es el header para marcar requerimientos de QoS, luego el bit indicado como R, $R = 0$ señala soft-RT y $R = 1$ hard-RT. La parte señalada como Flow Label es la 3-tupla $(bandwidth, buffer, delay)$ siendo 6 bits para BW, 5 bits para Buffering, y los últimos 5 bits para delay. Este último modelo es similar al propuesto en la tesis [Rob08] para combinar 802.11e con IPv6.

El documento tuvo 3 modificaciones, quedó solo como draft y no siguió teniendo discusiones, por lo tanto no alcanzó el estado de RFC Internet Standard Track. El autor, Rahul Banerjee, tiene varias propuestas de draft, aunque parece que ninguna alcanzó a ser RFC. Entre estas se pueden mencionar las siguientes:

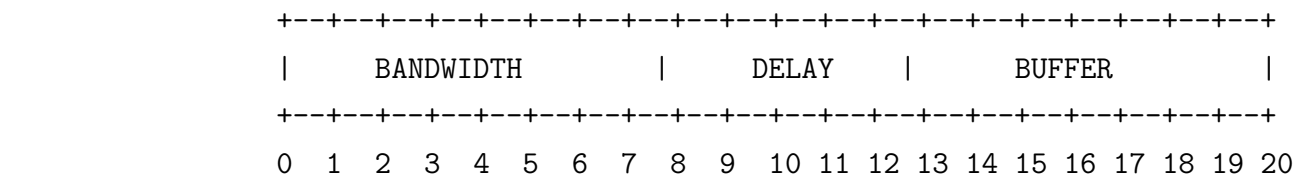
- "An extension of RTP and RTCP protocols For Video-on-Demand systems" (draft-banerjee-rtp-vod-00).
- "Impairment Constraints for Routing in All-Optical Networks" (draft-banerjee-routing-impairments-00).

Dentro de los documentos del autor la más relacionada con el trabajo se encuentra en: "Design and Implementation of the Quality-of-Service in IPv6 using the modified Hop-by-Hop Extension header - A Practicable Mechanism" (draft-banerjee-ipv6-quality-service-02) [BS02a], la cual propone un mecanismo de QoS Hop-by-Hop usando las cabeceras de extensión de IPv6 para marcado.

6.8. Un enfoque radical para proveer QoS sobre Internet usando el campo Flow Label de 20 bits

El draft "A Radical Approach in providing Quality-of-Service over the Internet using the 20-bit IPv6 Flow Label field" (draft-jagadeesan-rad-approach-service-01) [BS02b] tiene una importante base en el draft con muy poco tiempo de diferencia citado:[BSM02]. El mismo sugiere un enfoque radical y genérico para el Flow Label, pero parece ser más una simplificación del modelo anterior, salvo que no exige un formato, sino sugiere uno posible permitiendo variaciones. El objetivo es marcar las necesidades de recursos como ancho de banda y delay en el campo Flow Label.

El formato que propone es el siguiente:




```
Bandwidth = (2^B) * 32 (8 bits)
Delay (in decimal) = (2^B) * 4 nanoseconds (5 bits)
Buffer Requirements = (2^B) * 512 bytes (7 bits)
```

Los equipos intermedios deben negociar Flow States a partir del plano de control y esta información debe ser almacenada en el router. En los Flow States se deberá almacenar la información indicada a continuación:

```
1. Flow label, Source address --3-tupla
      and destination address.  Se pueden usar wildcards.

2. Forwarding treatment       --Como se trataran los paquetes que
                                 acorden el criterio anterior.

3. Flow statistics            --Contadores de tráfico

4. Flow time                  --El tiempo de vida que queda del estado
```

Un valor en 0 (cero) del Flow Label indica que no tiene una etiqueta de QoS. El valor 0 indica un tratamiento best-effort. Los valores de Flow Label cargados por el origen no deben ser modificados en el trayecto. En el documento se menciona que el uso de valores de Flow Labels cambiables en el camino no se recomienda debido a que se agrega una sobrecarga extra, por ser necesario nuevas re-negociaciones entre routers para mantener el significado original. Se propone que los hosts realicen una posible asignación estática y manual de códigos de flujos con aplicaciones, se sugiere, de forma práctica, usar el archivo UNIX: `/etc/qos.conf`.

Todos los modelos que incluyen Flow States se terminan pareciendo mucho a IntServ y requieren de un protocolo de señalización. Esta propuesta es un ejemplo. Probablemente las implementaciones sufran de los problemas de la arquitectura, como podría ser la falta de escalabilidad. También sucede que varios de los modelos que especifican un valor para Flow Label indicando los parámetros requeridos contradicen las especificaciones estándares de Internet. De cualquier forma los estándares de la IETF no han aportado mucho en el uso del campo y las propuestas son válidas.

6.9. Especificación del Flow Label de IPv6

"RFC-3697: IPv6 Flow Label Specification" [RCCD04] es el primer documento draft estándar IETF que trata específicamente el campo Flow Label. Si bien el RFC-2460 lo considera, varias cuestiones quedaban sin definir. El resultado obtenido en RFC-3697 comienza con los trabajos del draft: "IPv6 Flow Label Specification (draft-ietf-ipv6-flow-label-00.txt)" en Febrero de 2002. Luego este es reemplazado por otros hasta llegar a la versión final. En el mismo se brinda una definición del campo Flow Label que incluye las principales reglas pare el uso del mismo:

- Los nodos orígenes deben ser capaces de marcar con el campo los flujos de datos conocidos, habitualmente las conexiones TCP, incluso si no requiere un tratamiento especial.
- Si un nodo no provee tratamiento específico para los flujos debe ignorar el campo y transmitirlo como el tratamiento default ("best-effort").
- Si un nodo no asigna/identifica tráfico con flows debe configurar el campo a 0(cero). Los paquetes que no son identificados como parte de un flujo deben tener este valor.

- Para habilitar a las aplicaciones y los protocolos de transporte a definir que paquetes constituyen un flujo, el origen deberá proporcionar medios para que estas especifiquen que valores de Flow Labels son asociados con estos. Los valores de la etiqueta están sujeto a privilegios adecuados.

- Los valores de los campos de la 3-tupla: dir. origen, dir. destino, Flow Label identifican que paquetes pertenecen a un flujo.

- Los valores del campo en la 3-tupla: dir. origen, dir. destino, Flow Label; asignados no deben ser repetidos dentro de 120 segundos luego de la terminación del envío de los paquetes del flujo inicial. Se debería poder aplicar tiempos de cuarentena mayor por la aplicación.

- El origen debería asignar a conexiones de transporte/aplicación no relacionadas diferentes valores de Flow Label y debería ser capaz de solicitar valores no usados sin necesidad de indicar valores particulares.

- El origen debe evitar el re-uso en caso de que existan flows activos identificados con el valor. Para evitar re-uso accidental debería existir un mecanismo bien definido que permita asignarlos, podría ser de forma secuencial o mejor, (pseudo)-aleatoria. Se debería mantener a los valores en uso para evitar el problema ante inicios/re-inicios del sistema.

- El valor del Flow Label cargado por el origen debe ser transmitido sin cambios hasta el destino.

- Los nodos IPv6 no deben asumir ninguna propiedad matemática u otra sobre el valor del campo.

- La performance de los routers no debería depender de la distribución de los valores o de los valores mismos del campo. Indicando especialmente que no es por si solo una buena entrada para funciones de hashing.

Las cuestiones anteriores se aplican al manejo de flujo sin estado principalmente. El documento también define los requerimientos para el manejo con estados, habitualmente llamados Flow States. Para esto define:

- Los paquetes pueden ser manejados con un tratamiento especial de acuerdo al campo si se ha generado un Flow State específico.

- Para poder tener un tratamiento por flujo específico, se requieren establecer Flow States en los nodos que estarán en el camino. No necesariamente deben ser todos. Indica que el mecanismo será definido en una especificación separada.
- El método de manejo de Flow States debe permitir los medios para la limpieza del estado para el tratamiento de un flujo específico. Se debería poder indicar mediante una señalización tiempos mayores a 120 segundos para la vida del flujo.
- El establecimiento del estado debe ser capaz de recuperarse cuando los estados no son soportados. Un ejemplo podría ser enviando como "best-effort" con valor cero.
- No especifica el documento claramente como se crean, por lo tanto se asume que pueden ser dinámicos o estáticos.
- Los nodos intermedios que manejan estados no deben asumir que los paquetes que parecen pertenecer a un flujo arribando más tarde que 120 segundos sean del flujo salvo que explícitamente se haya expresado un tiempo de vida mayor en el proceso del establecimiento del estado.

Con respecto a cuestiones de seguridad menciona que el campo Flow Label no es cubierto por IPSec en modo transporte. En modo túnel se cubre el campo del paquete encapsulado, pero si este requiere que sea analizado hop-by-hop no sirve.

En conclusión este documento da los lineamientos para el uso del campo, dejando muchas propuestas, como las analizadas anteriormente, no compatibles con el estándar IETF. En 2011 el RFC-3697 es reemplazado por RFC-6437 intentando flexibilizar algunos aspectos, aunque no se parece tener los resultados esperados: difundir aplicaciones y el uso del campo.

6.10. Usando el campo Flow Label de 20 bits del encabezado IPv6 para indicar la QoS deseable para servicios en Internet

La tesis "Using the 20 bit flow label field in the IPv6 header to indicate desirable quality of service on the Internet" [Pra04] hace una propuesta de uso del campo Flow

Label para indicar los parámetros de QoS requeridos de forma detallada. Indica los siguientes valores con sus significados para el encabezado del campo:

```
Bit Pattern   Approach
-----------------------------------------------------------------
00            No QoS requirement (Default QoS value)
01            Pseudo-Random value used for the value of Flow-Label
10            Support for Direct Parametric Representation
1100          Support for the DiffServ Model
1101          Reserved for future use
111           Reserved for future use
```

Permite la asignación (pseudo)-aleatoria o con valores específicos. También define formatos para valores específicos, por ejemplo, one-way delay, bandwidth, pérdida (loss), etc. Se propone el siguiente formato:

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|1 |0 |R | OWD       | IPDV   | BW               |  OWPL     |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20

R: RT strict=1 (strict), RT soft=0 (available)
OWD: one-way delay
IPDV:  IP Delay Variation (Jitter)
BW: bandwidth
OWPL: one-way-packet-loss
```

Definiendo luego tablas que asocian valores binarios para campos de acuerdo a su significado. Según RFC-6294, la propuesta parece asumir que hay un RSVP por detrás como RFC-2205 [BZB+97] que implementará los parámetros.

En el articulo no implementa nada ni muestra resultados que justifiquen su implementación aunque reclama que la propuesta es simple, escalable, modular y genérica. No parece ser así, pues no hay desarrollos.

6.11. Comparación del rendimiento de la QoS en IPv6 entre los modelos de IntServ, DiffServ

En "Comparison of QoS Performance between IPv6 QoS Management Model and IntServ and DiffServ QoS Models" [FDP+05] de El-Bahlul Fgee y otros autores, se compara mediante el uso del simulador NS/NS-2 [TOLa] modelos para proveer QoS. El artículo menciona cuatro esquemas para QoS:

- IntServ
- DiffServ
- IPv6 QoS
- MPLS

Se indica en el artículo además otro modelo, que es el actual en Internet, best-effort. Los modelos se describen de la siguiente forma:

Best Effort: Modelo actual de QoS en Internet, "Mejor Esfuerzo", todo el tráfico es tratado de la misma forma, no hay garantías de parámetros de QoS. Su desventaja es que no provee directamente QoS y su ventaja es la simpleza.

IntServ: [BCS94] Simula un sistema de circuitos sobre IP. Utiliza RSVP (Resource Reservation Protocol) para señalización del camino por donde se requiere la QoS. Se requieren mensajes de refresco para mantener el camino. RSVP genera "Soft States" en los routers, estos pueden ser modificados por nuevos mensajes. Trabaja con un esquema per-flow QoS, trata de cubrir los requerimientos de QoS por flujo de datos.

Menciona como desventajas:

- Carencia de Escalabilidad. Demasiados recursos de procesamiento y almacenamiento son requeridos por los Soft States. Los flujos generan una granularidad muy fina lo que produce que no escale el modelo. Solo aplicable a redes chicas.
- "Todos" los routers deberían implementar RSVP, extra al tratamiento de la QoS, control de admisión, clasificación, etc.

DiffServ: [BBC+98] Modelo más simple que IntServ. En este el tráfico entra en la red (dominio de QoS) y es clasificado en los bordes, se agrupan diferentes flujos en clases más amplias, PHB (Per-hop-behaviors). Granularidad gruesa. Podría utilizar QoS/Bandwidth Brokers para administrar y negociar los requerimientos, comunicar con los routers de borde (edge routers) y tracear las reservas de recursos o podría implementarse de forma estática. El tráfico, una vez marcado, es tratado dentro del dominio de acuerdo a su marca. Todos los paquetes de la misma clase deberían ser tratados de la misma forma dentro del dominio de QoS. DiffServ define algunos tratamientos que deberían aplicarse en cada nodo a lo largo del camino de acuerdo a la clase. Básicamente se utiliza el campo DSCP (Differentiated Code Point) en IPv4 o Traffic Class en IPv6. Es un modelo más sencillo y más escalable que IntServ. Desventajas:

- Se agrupan flujos individuales en una misma clase, no pueden ser diferenciados. Pocas clases.
- Modelo más estático en la implementación.
- Puede producirse un delay mayor en el mapeo a las clases.
- No se adecúa correctamente a un modelo de extremo a extremo.

Otros modelos de QoS complementarios que se han desarrollado recientemente:

MPLS (Multi Protocol Label Switching): [RVC01] puede utilizar el modelo de DiffServ o IntServ. Ofrece la posibilidad de trabajar con Ingeniería de Tráfico. Modelo más eficiente, extensible a IPv6. No se indican más detalles en el artículo.

Por último, analiza:

Modelo de IPv6 QoS: Brevemente su funcionamiento es descripto así: antes de transmitir información, se debe enviar el requerimiento de QoS. Este lo recibe uno de los routers de borde y lo debe enviar al Controlador del Dominio de QoS, quién lo aprueba o no. El resultado lo comunica el router de borde al emisor. Si se acepta se comienza a enviar. El requerimiento lleva los campos: tiempo (duración), tipo de servicio (rate promedio, burst y pico), el DGI y la dirección destino. Se utiliza el Domain Global ID (DGI) o Packet ID: (dirección origen IPv6 + Flow Label) para reservar y mantener track de los flujos en lugar de usar la 3-tupla vista en la mayoría de los documentos relacionados.
El modelo indica que el tráfico se clasifica en los routers de borde y es tratado

con la QoS adecuada, previamente configurada por el Controlador de QoS. Es usado el campo TC para indicar el tratamiento diferenciado. Los router en las simulaciones utilizan WFQ, y se calcula el peso de acuerdo al campo TC (Traffic Class), usado como prioridad. En las publicaciones no hace referencia a como el TC se determina (configura). Se indican como ventajas del modelo:

- No se negocia la QoS con el emisor. El router de borde lo hace con el QoS manager/controlador.
- El modelo dice ser más sencillo de implementar porque no requiere protocolos externos de señalización.
- El QoS manager o Controlador de QoS toma ventaja de los campos Flow Label y Traffic Class para reservar y llevar la traza de la utilización de recursos de la red. Los beneficios de usar el Flow Label son: procesamiento más rápido, solo campos fijos de la cabecera. Evita *layer violation*, ya que no se debe "meter" dentro de PDU de las capas superiores, como podría ser en la capa de transporte mirando puertos u otros datos. Este tema ya es considerado y analizado mejor en RFCs.

Su funcionamiento no es descripto en mucho detalle en la publicación, y, a igual que en otras, solo es una implementación basada en el simulador NS-2. No se tiene en cuenta la escalabilidad ya que solo habla de un controlador genérico y deja muchos detalles importantes sin especificar. Se indica que no requiere un protocolo de señalización externo, pero no describe como se trabaja con los mensajes de control que incluye en su descripción, como son aquellos entre los routers de borde y el controlador.

En artículos anteriores como "Implementing QoS capabilities in IPv6 networks and comparison with MPLS and RSVP"[FPRS03] o "Implementing an IPv6 QoS management scheme using flow label & class of service fields"[FKR+04] sienta la base para este artículo. El primero presenta un esquema para garantizar la QoS para tráfico de Real Time y mejorar el uso de los recursos del backbone minimizando el delay. Las simulaciones son realizadas sobre el simulador NS y compara RSVP, MPLS, IPv6 con WFQ e IPv6 con CBQ. Mide end-to-end delay, rendimiento y pérdida. La diferencia parece hacerla CBQ sin importar si es MPLS, RSVP o IPv6. Usa para IPv6 el campo Flow Label para hacer un fast-switching y el campo TC para dar prioridad.

El artículo de 2005 es bastante superficial y no aclara detalles, sirve como base de clasificación para Modelos de QoS, aunque el caso particular de IPv6 es uno definido ad-hoc, que no está acorde en algunos puntos con los RFCs. Apunta a un modelo con estados.

6.12. Clasificación de paquetes IPv6 basada en el Flow Label, dirección origen y destino

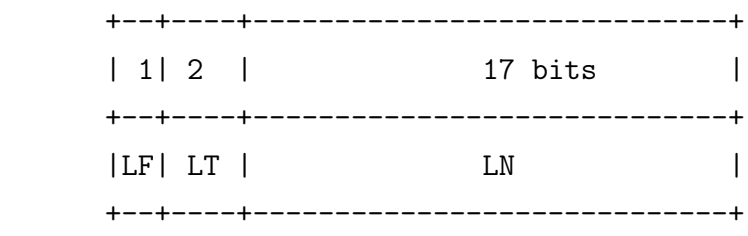
El artículo de 2005 "IPv6 Packet Classification Based on Flow Label, Source and Destination Addresses"[PE05] muestra una implementación para los clasificadores de paquetes según la 3-tupla: Flow Label, Dir. Origen, Dir. Destino. Utiliza una estructura de datos llamada Hierarchical-Trie (H-Trie), indicando que es adecuada y permite obtener un buen rendimiento para los hashes que se generan. El desarrollo del algoritmo está realizado sobre el lenguaje Java en una plataforma Windows de un solo thread.

La clasificación de tráfico es importante para ofrecer un menor delay y un mejor tratamiento en el curso del tráfico. En el caso de IPv4 para llevarla a cabo se requiere leer campos de capas superiores, por ejemplo, puertos TCP/UDP. Esto genera problemas ante la fragmentación o tunelización con cifrado.

El artículo muestra otra ramificación del estudio y aplicación del campo FL de IPv6. En este caso algoritmos para clasificar y lograr una mejor distribución en un espacio de hashing.

6.13. Proveyendo QoS de extremo a extremo usando el Flow Label de IPv6

En el documento "End-to-End QoS Provisioning by Flow Label in IPv6"[LTH06] se especifica un formato de Label donde 17 bits se generan de forma (pseudo)-aleatoria.




```
LF   0  Disable
     1  Enable
LT  00  Flow label requested by source
    01  Flow label returned by destination
    10  Flow label for data delivery
    11  Flow label terminates connection
LN      Random number created by source
```

El flag/bit LF indica que el Flow Label tiene el significado del formato presentado anteriormente. La parte LT, de 2, bits indica quién solicitó el recurso y por último los 17 bits. Este marcado es incompatible con RFC-3697, ya que, se podría marcar este bit en 1 (uno) por una fuente que no utiliza esta definición, sin tener en este caso el significado esperado. Se trata de poner el control de este marcado en dispositivos que cumplan el rol de gateways de entrada. Los gateways parecen convertirse un poco en "cuello de botella" de la posible implementación.

En el documento se describe un mecanismo de señalización dentro de los mismos mensajes de datos entre el origen, routers y el destino donde se usa el campo Traffic Class y los bits señalizados anteriormente del Flow Label.

El funcionamiento es el siguiente: el origen marca los mensajes con $LF = 1$, $LT = 00$ y el resto con un valor (pseudo)-aleatorio, luego los debe enviar a un gateway que hará los chequeos que el sistema requiere (como el control de que no se repitan los valores). Este marcara el campo Traffic Class enviándolo luego al verdadero router de borde del dominio con QoS. El router examinará el Flow Label y enviará el paquete de acuerdo a los campos indicados. Cada paso generará un estado en los equipos intermedios que se mantendrá para dar un tratamiento especial a cada flujo. Al llegar al destino final, este responderá con determinada configuración en los campos del Flow Label para habilitar/establecer el camino en los equipos intermedios. Para dar de baja la reserva de recursos se envía un mensaje con los valores $LF = 1$, $LT = 11$ y el mismo

valor inicial para LN. Los gateway parecen ser los que tienen la responsabilidad de decidir el tratamiento de los mensajes, a partir del valor del campo Traffic Class. No se define como los hosts le indican este tratamiento a los gateways. Los gateways hacen la elección en base a la 3-tupla que identifica el Flow-ID como se vio hasta ahora. Usan una tabla que asocia el valor a un ToS de acuerdo al campo Traffic Class. Establecido el camino, los routers para definir el próximo salto en lugar de mirar la tabla de routing, miran una tabla que asocia el LN con el próximo salto. Las pruebas son llevadas a cabo usando el simulador NS-2 y parece que la ventaja, más que la implementación de QoS, es el Fast-Switching basado en la porción del Flow Label LN en lugar de utilizar la dirección IP destino.

6.14. Provisión de QoS en IPv6 de extremo a extremo en redes heterogéneas usando el Flow Label

En "IPv6 End-to-End QoS Provision In Heterogeneous Networks Using Flow Label" [SHM08] se evalúa sobre el simulador OPNET [TOLb] una propuesta para usar el campo Flow Label para ofrecer QoS end-to-end. Se propone una modificación a la especificación del mismo indicando que así el manejo de los diferentes tráficos puede ser más eficiente y se genera un nuevo modelo de QoS: "FL model". Se afirma que tiene una granularidad más fina y los resultados sobre el simulador muestran un mejor rendimiento que los existentes. Se propone un modelo en el cual el Flow Label puede ser cambiado en el ruteo pero manteniendo el significado original. Se podría ver como una aplicación híbrida del campo, donde se usa de forma end-to-end pero se puede cambiar cuando se pasa de un dominio de QoS a otro.

El documento afirma que mantener el valor del Label sin cambiar trae problemas de escalabilidad y sincronización en un entorno multi-dominio como Internet. La propuesta es bastante complicada donde se incluyen nodos con diferentes funciones como FLB (Flow Label Broker) para negociar el valor del campo dentro del dominio incluyendo la política a aplicar, y QoS Gateways que mantendrán los mapeos de valores del campo del dominio de donde proviene al nuevo. Los FLB serían reemplazos de BB (Bandwidth Brokers). Se indica como las principales características del modelo:

- End-to-end QoS puede ser garantizada. La interpretación de las clases de QoS es realizada por flujo, dando una mejor granularidad.

- No se requieren modificaciones a los modelos/tecnologías de QoS actuales. Los únicos cambios son los FLB y los QoS gateways en cada dominio.
- La clasificación de los paquetes se hace solo en una dimensión del campo Flow Label, dando una mejor velocidad de procesamiento.

De los anteriores, el punto dos indica que no se requieren modificaciones, pero en realidad los QoS Gateways serán los routers de límite/borde que deberán hacer un procesamiento diferente, como es la comunicación con los FLB. Los QoS Gateways se muestran en el documento como un eslabón más en la cadena, como previos a los routers del sistema/dominio de QoS. El modelo parece tender a generar estados similar como sucede con IntServ [BCS94], por lo tanto se pone en duda la escalabilidad del mismo. También se define un proceso de señalización: FL-SIP, el cual es usado por el origen contra el FLB de su dominio, entre los FLB de diferentes dominios y entre los FLB de entrada y los QoS Gateways del mismo dominio. En el artículo se realizan comparaciones sobre una maqueta contra un modelo DiffServ. En el mismo varios detalles no quedan claros, por ejemplo, la compleja comunicación entre los participantes mediante el protocolo de control que se menciona. Ente los puntos interesante se puede ver que hace una breve revisión de los trabajos relacionados, aunque en muchos casos no acierta en lo que expresan, por ejemplo, indica que la referencia [BSM02], hace una propuesta de etiquetas con solo significado local, y en realidad no es el objetivo dentro del documento que se analizó anteriormente: "A Modified Specification for use of the IPv6 Flow Label for providing An efficient Quality of Service using a hybrid approach" [BSM02].

6.15. Provisión de QoS en IPv6 de extremo a extremo en redes heterogéneas usando agregación del Flow Label

"IPv6 End-to-End QoS Provision in Heterogeneous Networks Using Aggregated Flow Label" [SHM09] es una extensión del artículo anterior citado como [SHM08]. En este se propone agrupar/agregar flujos que requieran una misma QoS, de forma similar como pasa con DiffServ de acuerdo al valor del DSCP/TC. El modelo es llamado FL-AG (Flow Label Aggregate). Diferentes flujos con distintas direcciones podrían ser asignados al mismo FL y de esta forma obtener una granularidad más amplia.

El modelo usa dos tipos de agentes o entidades: el FLB (FL Broker) encargado de negociar el QoS-CT (QoS Class table) y el valor agregado del Flow Label. Tienen una lista de FL-ST (FL state table), con los FL asignados a c/u y los tratamientos de QoS indicados en la QoS-CT. Luego los QoS Gateways se encargan de negociar los mapping entre los diferentes dominios de QoS por donde atravesará el tráfico.

El proceso consta de 3 fases, una primera para generar el valor FL de forma off-line, es decir asociar el tratamiento con los valores de los flujos; la segunda para es el "FLB negotiation process", que dado el requerimiento se le asigna el valor el campo con el cual debe ser marcado el tráfico y por último la baja, "FL deletion" llevada a cabo por FLB y QoS gateways ante la expiración de la información para el FL (Flow). Los resultados son mostrados con el simulador OPNET.

6.16. Estudio de casos de uso propuestos para el Flow Label de IPv6

En el RFC informativo "Survey of Proposed Use Cases for the IPv6 Flow Label" [HB11] se realiza una recopilación y análisis de varias propuestas de uso del campo Flow Label en IPv6. Este documento luego es citado por RFC-6436, que también se analiza en otra sección. Entre los trabajos mencionados podemos indicar los siguientes: cita de [MH00], los trabajos de A. Conta [CR01a], [CR01b], y de otros autores, como [Hag01] de Hagino.
Los documentos analizados se encuentran catalogados en categorías que definen el mismo RFC con respecto a su uso. De acuerdo a su categoría son:

- Para QoS End-to-end QoS, usando valor (pseudo)-aleatorio de Flow Label. Analiza el documento [LTH06] ya analizado en este mismo texto.
- Para Load-Balancing, usando valor (pseudo)-aleatorio de Flow Label. Referencia al trabajo en progreso, que luego termina en RFC-6438 [AC11]. Este es compatible con las reglas de RFC-3697.
- Como Security Nonce, usando valor (pseudo)-aleatorio. Analiza el draft [Bla09] de Blake. Esta propuesta no requiere trabajar con estados y permite una defensa contra ataques de SYN/ACK generando valores (pseudo)-aleatorios para el campo haciendo los encabezados más difícil de predecir por el atacante. Cumple con las "reglamentaciones" IETF.

- Especificando Parámetros/Requerimientos de QoS en el campo. Analiza [Pra04] y [LK04]. El primero ya es analizado en este documento. En el segundo, se define un formato similar a los anteriores, con 5 campos donde 2 son encabezado de interpretación y en los restantes se expresan valores para delay, bandwidth y buffer. Se proponen, además, monitores de rendimiento y la posibilidad de reserva de capacidades de forma dinámica. El esquema sirve para IntServ como para DiffServ. Los paquetes que no utilizan los valores indicados para el propósito de QoS, por ejemplo, con valores (pseudo)-aleatorio, son finalmente tratados como "best-effort". Las pruebas se realizan sobre el simulador QoSSim[TOLc], que se desarrolla específicamente para este propósito. Parece que ninguna de las dos propuestas son compatibles con las reglas de IETF, RFC-3697, debido a los encabezados codificados que se agregan con significado específico y particular.

- Usados para el control del switching/forwarding hop-by-hop. El primer draft comienza en 2005 y la última modificación es de 2008 [CBB08]. En estos se describe y propone una arquitectura llamada 6LSA: IPv6 Label Switching Arquitecture donde se utiliza el campo para hacer un fast-switching muy similar como sucede con los FEC en MPLS [RVC01]. Define un protocolo de intercambio de Labels. Una parte del Label será una etiqueta con significado local que se mantendrá a lo lago del LSP (Label Switch Path) y la otra local para hacer el fast switching. La propuesta requiere re-escribir el valor hop-by-hop lo cual la hace incompatible con las reglas de IETF RFC-3697. Existen otras propuestas con el objetivo de acelerar el look-up en el switching, cuestión que parece haber tenido mayor relevancia en el momento de la concepción de IPv6 y sus direcciones "extra-largas" de 128 bits. Hoy con el avance de la tecnología esta limitación ha sido superada.

- Usado para especificar extensiones a la arquitectura DiffServ. Aquí analiza [BSM02], también revisado en este texto. El método rompe con las reglas de RFC-3697 y no es compatible.

- Otros usos. [DE11] propone usar en el Dual-Stack Lite el campo para distinguir el encapsulamiento IPv4-in-IPv6 y la traslación de direcciones realizada. Deriva el valor del campo a partir de la 5-tupla que incluye las direcciones, números de puerto y el protocolo, Next-Header. Indica que los valores generados deben ser únicos y que pueden ser usados para priorizar tráfico. El método parece ser compatible con las RFC de los estándares IETF. También se encuentran

referencias a “RPL: IPv6 Routing Protocol for Low power and Lossy Network” y a Mobile IP con MEXT (Mobility EXTensions for IPv6).

El documento indica que los avances en tecnologías de hardware de high-speed-switching y el éxito de MPLS permiten descartar el uso del Flow Label considerado para implementar un rápido procesamiento en los equipos intermedios, conocido como fast-switching. De la misma forma menciona que la arquitectura de DiffServ es vista como suficiente para codificar la QoS. A pesar de esto gran parte de las propuestas sobre el campo asumen que su uso es para QoS, algo que a menudo se requiere, pensando en la necesidad de la “Network Neutrality”.

También marca que la violación del esquema en capas de acuerdo a donde se ubica la labor del agente/dispositivo que procesa el paquete no es algo que sea realmente una preocupación académica, aunque este también es un tema abordado por los diseñadores. El uso del campo para hacer una mejor y más rápida clasificación sin romper las fronteras de las capas del modelo TCP/IP u OSI es un punto a favor.

Enumera las tres propiedades controversiales de RFC-3697, tratadas en RFC-6436, marcando que fue una decisión adrede debido a la semántica sin estado y el end-to-end que debe tener IP.

Finalmente aconseja a que deben enfocar los investigadores y diseñadores en temas relacionados respetando las reglamentaciones/recomendaciones IETF. Una posibilidad es usarlo como nonce(valor numérico) para generar valores hash, pero nunca de forma aislada. Otra forma es marcando tráfico que pertenece a una misma sesión, pero sin usar el valor para influir en las decisiones de ruteo. En el caso de querer utilizarlos directamente para QoS, recomienda ignorar las reglas de la RFC, pero que el alcance de la implementación solo sea dentro de un dominio determinado (conjunto de nodos y routers), por ejemplo, un AS, pero sin permitir que el comportamiento se propague fuera. Esto se podría implementar de forma que, cuando el paquete sale fuera del dominio, los routers de borde re-escriban el valor a cero u otro pactado respetando la RFC. De cualquier forma, la mayor cantidad de las propuestas generadas son incompatibles con la especificación RFC-3697 y trabajos del RPL o MEXT ponen en cuestión el mismo documento.

6.17. Justificación de la Actualización de la especificación de IPv6 Flow Label

El documento "Rationale for Update to the IPv6 Flow Label Specification" [AJC11], RFC de categoría informativa, no de estandarización de protocolos, hace un estudio de los inconvenientes que se encuentran en RFC-3697 y sienta la base para el nuevo draft estándar, RFC-6437. En el mismo se analizan los siguientes tres puntos cuestionados del RFC anterior.

- El valor del Flow Label cargado por el origen debe ser transmitido sin cambios hasta el destino.
- Los nodos IPv6 no deben asumir ninguna propiedad matemática u otra sobre el valor del campo.
- La performance del router no debería depender de la distribución de los valores o de los valores mismos del campo. Indicando especialmente que no es por si solo una buena entrada para funciones de hashing.

Adicionalmente, indica que en el RFC-3697 no se definen métodos para seleccionar los valores del campo y que la acción se delega en parte a la señalización, pero no se encuentran actualmente métodos estandarizados para hacerlo, salvo RFC-2205, documento que parece no tener mucha vigencia, ya que hace referencia al campo como de 24 bits (versiones previas al estándar IPv6). El RFC-2205, es corregido por el RFC-6437 con respecto a la longitud del campo.

En el documento analizado en este párrafo se menciona la falta de uso del campo en la práctica y analiza las 3 propiedades anteriores. Para la primera, las consecuencias son que los sistemas intermedios no pueden re-escribirlo ni siquiera por razones de seguridad. También indica que si se re-escribe, el valor no podría ser confiable, pues no hay mecanismo de integridad del mismo, como sucede en el caso de usarse para QoS. Se apunta que esta propiedad debería ser relajada. La QoS no es muy aceptada en el documento debido a dicha característica.

Para la segunda indica que las palabras "no debe asumir" no prohíben usarlo para diferentes tratamiento de los paquetes. Si un nodo intermedio, un router, conoce por configuración o por protocolo de señalización como tratar determinado flujo con determinado valor lo puede hacer usando este conocimiento obtenido por otro

medio. Simplemente aclara que se debe prestar atención para no generar posibles mal-interpretaciones.

Para la última regla marca que el término "solo" no implica poder combinarlo con otros campos para generar una buena entrada para el hashing. Recomienda que no se use de forma aislada, pero sí podría hacerse de forma combinada.

Se menciona que varias propuestas y algunas implementaciones, de acuerdo a la semántica que requieren darle al campo, terminan rompiendo las reglas "impuestas". Por ejemplo, requieren:

- Los equipos intermedios cambien el valor.
- El receptor a menudo no usa el campo y debería hacerlo.
- Parte de los bits del campo necesitan una codificación con significado particular que será usado por los equipos intermedios.
- Las codificaciones del campo marcan QoS e incluso identifican el flow.
- El valor del campo es usado para que los equipos tomen decisiones en el proceso de forwarding/switching.

No se recomienda el uso de valores secuenciales y aconseja valores (pseudo)-aleatorios.

Como dato concluyente indica que varios investigadores y diseñadores han sido coartados, inhibidos, por las reglas de RFC-3697, aunque también existen propuestas que cumplen total o mayormente con la especificación. Afirma que se podría pensar en relajar algunas cuestiones pero tratando de seguir los lineamientos del RFC original, cuestión que se aborda en el RFC-6437.

6.18. Especificación del Flow Label de IPv6 (nueva versión)

"RFC-6437: IPv6 Flow Label Specification" en su versión nueva [AJCR11], que deja obsoleta la anterior, RFC-3697. En el mismo documento se puntualizan cuales son las principales diferencias con su antecesor.

- Recomienda el uso de valores distinto de cero e indica los mecanismos/procedimientos para asignar los valores.
- Incentiva el uso de un modelo sin estado y alienta al uso de valores para el campo que estén uniformemente distribuidos, apuntando a alto grado de variabilidad.
- Debido a que incentiva el uso de modelo sin estados, no especifica ningún detalle para hacerlo con uno stateful (con estados). No lo prohíbe, pero no da detalles. El caso de las restricciones de tiempo de 120 segundo o mayores son suprimidas.
- Retiene la regla que indica que el valor debe ser marcado por el origen y no debe ser modificado, pero permite la flexibilidad de que en el caso de que el origen no lo asigne, los routers pueden hacerlo en nombre de estos.
- La especificación tiene en cuenta la posibilidad del uso del campo como Covert Channel, es decir, mandar información en forma de ataque de seguridad en el campo, que supuestamente no debería pasar por la política de seguridad. En el documento se indica la posibilidad de resetear por los firewall el valor a cero para mitigar este tipo de ataque.

No indica nada sobre manejo de estados y la capacidad de tener tratamiento diferenciado de acuerdo a la información mantenida, permitiendo el uso de estos, posiblemente con un protocolo de señalización de por medio. A favor de los modelos sin estados recomienda que las implementaciones que hagan uso de los estados no interfieran con los modelos stateless. Corrige la longitud del campo del RFC-2205 de 24 a 20 bits e indica que esta RFC sigue vigente con su correspondiente modificación.

Luego mantiene las siguientes reglas:

- Los nodos orígenes deben ser capaz de marcar con el campo los flujos de datos conocidos, tal el caso de conexiones TCP, aun si no requieren un tratamiento especial.
- Si un nodo no asigna/identifica tráfico con flows debe configurar el campo a 0(cero). Los paquetes que no son identificados como parte de un flujo deben tener este valor.
- Los valores de los campos de la 3-tupla: dir. origen, dir. destino, Flow Label; identifican que paquetes pertenecen a un flujo.

- Los routers/nodos de forwarding/load balancers (balanceadores de carga) no deben depender solamente de los valores del campo, se debe combinar al menos con otros campos.

También hace un análisis más exhaustivo del punto de vista de la seguridad, indicando que el valor podría ser modificado por cualquier nodo intermedio con el objetivo de generar un DoS (Denial of Service), o algún otro tipo de ataque. Indica que el uso para QoS podría verse afectado con un ataque, derivando en un tratamiento del tráfico todavía peor que "best-effort".

6.19. Usando el Flow Label de IPv6 para balance de carga en "granjas" de servidores

El artículo "Using the IPv6 Flow Label for Load Balancing in Server Farms" RFC-7098 [CJT14], parece ser uno de los últimos RFC para la fecha de este texto que trata el uso del campo de IPv6. Este sigue la línea del IETF donde se apunta al balance de carga.

6.20. Conclusiones de los documentos analizados

Existen numerosos documentos tratando el uso del campo Flow Label de IPv6 para brindar una mejor QoS. Por ejemplo, artículos anteriores a la RFC de IPv6, con protocolos para QoS con Flow Label como ImpRes [BMFM96]. Otros textos, no analizados en este, tratan de aplicar el modelo InServ a IPv6, como IntServ6[PHPH05] o [WZDW09] que combina IntServ con SNMP para QoS sobre IPv6.

Muchos de los textos que se estudiaron apuntan a un modelo de IPv6 con DiffServ y/o fast-switching. Con las últimas modificaciones del RFC los modelos sin estados son los que se recomiendan aunque tienden a limitar los posibles desarrollos para el uso del campo. Los modelos con estados, si bien no se fomentan, no son prohibidos.

Todos los artículos que realizan comparaciones se basan en simuladores y no aparece ninguno que implementa un modelo, total o parcial sobre sistemas operativos completos. La mayoría de los artículos que usan IPv6 con QoS mencionan componentes accesorias como QoS Gateways, QoS Brokers, FL Brokers protocolos de señalización que están vagamente especificadas y que se olvidan al momento de declarar que

el modelo de QoS de IPv6 es más sencillo que IntServ o DiffServ.

La falta del uso del campo Flow Label en IPv6 sigue siendo una realidad y la aplicación del mismo para QoS parece haber perdido la atención de la IETF, aunque no por parte de otros investigadores.

Capítulo 7

Flow Label en práctica

7.1. Introducción

Para hacer uso del nuevo campo **Flow Label** agregado por IPv6 es necesario que existan mecanismos bien definidos para que los nodos de la red puedan marcarlo, remarcarlo y hacer el "matching" (compararlo) para la clasificación. Este aspecto tiene que ver exclusivamente con las implementaciones, el desarrollo y la difusión que las mismas tengan. En el capítulo se analizarán las posibilidades existentes de implementaciones y algunos lineamientos informativos dados por la IETF mediante documentos RFC. Se brinda como referencia bibliografía que presenta los detalles de la programación de IPv6 los libros citados a continuación: [SFR03] y [iJi04].

Cuando se habla de interfaces de programación de red, existe la ampliamente difundida API de socket incluida por BSD, hoy presente en la mayoría de los sistemas y estandarizada por POSIX. Desde el punto de vista de los nodos finales se estudiarán las alternativas que esta interfaz de programación ofrece para el marcado. Con respecto al manejo por los nodos intermedios, i.e. routers, el tratamiento no es estandarizado, debido a que la gran parte de este procesamiento se realiza en los kernels. Hoy existe una gran proliferación de sistemas basados en kernel GNU/Linux, por lo que, los que estén cimentados en el mismo compartirán probablemente el manejo que este ofrezca. Otras alternativas son los sistemas BSD. Estos parecen no haber prestado mayor atención al uso del campo por el usuario debido a la falta de definición y consenso sobre la utilidad del mismo por la IETF.

7.2. Análisis de lineamientos de IETF para implementación

Los trabajos de IETF sobre la definición del API para trabajar con IPv6 arranca en 1995 con el draft: "IPv6 Program Interfaces for BSD Systems": draft-ietf-ipngwg-bsd-api-00. [GTB95]. Luego deriva en el primer estándar: RFC-2133: "Basic Socket Interface Extensions for IPv6" [GTBS97] el cual es más tarde obsoleto por RFC-2553 [GTBS99], que luego queda superado por RFC-3493 [GTB+03]. En estos textos se dan definiciones informativas de las características que debería tener el API de socket para IPv6, de forma de ser lo menos problemática al introducir cambios y tratar de mantener la compatibilidad hacia atrás. La estructura de `socketaddr` finalmente definida es la que se muestra en el código a continuación.

Listado 7.1: Socket address struct IPv6 según RFC-3493

```
struct sockaddr_in6 {
    sa_family_t     sin6_family;    /* AF_INET6 16bits */
    in_port_t       sin6_port;      /* transport layer port # */
    uint32_t        sin6_flowinfo;  /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id;  /* set of interfaces
                                       for a scope */
};
```

7.2.1. RFC-2133: Basic Socket Interface Extensions for IPv6 (Obsoleta)

En RFC-2133 se da la primera definición informativa del API. En el mismo se describen las nuevas características ofrecidas por IPv6 destacando el campo prioridad y el campo Flow Label. El campo prioridad, definido como **priority** es el que más tarde será llamado **Traffic Class**. Se indica la necesidad del cambio de la API de socket para IPv4 para poder brindar acceso a estos campos además del hop-limit y la inclusión de nuevas opciones. Se indica la necesidad del cambio del API también para el envío y recepción de tráfico multicast.
Debido a la fecha del documento en el mismo se considera que la longitud del campo en el datagrama es de 24 bits en lugar de la longitud finalmente estandarizada de 20 bits. Se hace referencia a la estructura de datos para el socket con el campo `sin6_flowinfo` de 32 bits donde los primeros 24 son para el **Flow Label** y 4 bits para el campo prioridad. El contenido y la interpretación de estos valores queda sin

especificar en el documento como explícitamente lo menciona: "...The contents and interpretation of this member is unspecified at this time...".

7.2.2. RFC-2553: Basic Socket Interface Extensions for IPv6 (Obsoleta)

En 1999 el documento RFC-2553 deja obsoleto a RFC-2133. En el mismo ya se hace referencia al **Flow Label** como un campo de 20 bits, y en lugar de llamar **priority** al campo usado por DiffServ lo llama **Traffic Class**. En el documento se indica al campo `sin6_flowinfo`, como estructura de datos de programación, también de 32 bits. Este contiene dos porciones de información, una para el valor de **Traffic Class** y la otra para el **Flow Label**. En el documento se indica que la interpretación de los mismos debe ser acorde con el RFC-2460. El valor del campo `sin6_flowinfo` debe ser colocado en cero previo al uso de la estructura por las aplicaciones al momento de la recepción. En RFC-2460 la interpretación no es clara por lo que el documento tampoco da claridad sobre el mismo.

7.2.3. RFC-3493: Basic Socket Interface Extensions for IPv6

Luego, en 2003, surge un nuevo documento informativo dando cambios para las especificaciones del API de programación de IPv6. De la misma forma que el documento anterior, RFC-2553, indica que el campo `sin6_flowinfo` de 32 bits, que tiene como intención contener la información de flow, no tiene una especificación exacta de como mapearlo con la información llevada por el datagrama IPv6: "...The sin6_flowinfo field is a 32-bit field intended to contain flow-related information. The exact way this field is mapped to or from a packet is not currently specified...". Hasta el momento de la definición del RFC, indica que el campo debería llevar el valor cero al construir la estructura `sockaddr_in6` y ser ignorado cuando se lee. Todos los valores de los campos en la estructura de dirección del socket obtenidos mediante la llamada `getaddrinfo()` que no son completados con argumentos explícitos deberían ser seteados a cero, por ejemplo, el caso de `sin6_flowinfo`.

Como cambio con respecto a RFC-2553 se plantea que en la versión previa el campo `sin6_flowinfo` es asociado con las partes del datagrama **Traffic Class** y **Flow Label** aunque su utilización no esa bien definida. En el caso del nuevo documento la definición completa queda diferida para el futuro: "...The complete definition of the sin6_flowinfo field, including its association with the traffic class or flow label, is now

deferred to a future specification...".

Con respecto a las modificaciones específicas sobre el API de sockets se menciona que los cambios se deben producir en los siguientes sectores:

- Funciones principales de socket.
- Estructura de datos, a la nueva definición: `struct sockaddr_in6`.
- Funciones de traslación de nombres a direcciones.
- Funciones de conversión de direcciones.

Por ejemplo, ahora la sintaxis para la creación de un socket TCP o UDP en IPv6 sería la siguiente:

Listado 7.2: Snippet de ejemplo para creación de socket IPv6

```
s = socket(AF_INET6, SOCK_STREAM, 0);
s = socket(AF_INET6, SOCK_DGRAM, 0);
```

Las funciones siguientes deben considerar como argumento el uso del nuevo protocolo.

Listado 7.3: Funciones con socket address como parámetros

```
bind()
connect()
sendmsg()
sendto()
```

Las funciones que retornan valores de direcciones deben considerar también el uso del nuevo protocolo.

Listado 7.4: Funciones con socket address como salida

```
accept()
recvfrom()
recvmsg()
getpeername()
getsockname()
```

No existen cambios con respecto a la sintaxis ya que todas las funciones anteriores usan punteros a direcciones opacas permitiendo trabajar sobre diferentes tipos de sockets, como RAW, TCP/IP, UNIX, etc.
A modo ilustrativo, la sintaxis definida en el documento para crear un servicio en el port 23 sería similar a la que se muestra a continuación:

Listado 7.5: Sintaxis de ejemplo de socket IPv6

```
struct sockaddr_in6 sin6;
        . . .
       sin6.sin6_family = AF_INET6;
       sin6.sin6_flowinfo = 0;
       sin6.sin6_port = htons(23);
       sin6.sin6_addr = in6addr_any;  /* structure assignment */
       /* sin6.sin6_addr = IN6ADDR_ANY_INIT; will NOT compile */
        . . .
       if (bind(s, (struct sockaddr *) &sin6, sizeof(sin6)) == -1)
```

Los campos para la estructura compatible con sistemas basados en 4.3BSD son:

sin6_family: identifica la "familia" de la estructura. Se coloca por encima del campo genérico `sa_family`. El valor debe ser **AF_INET6** definido en `<sys/socket.h>` Usualmente se define igual que **PF_INET6**.

sin6_flowinfo: campo de 32 bits que tiene como intención llevar el valor de flow, pero su definición y uso no quedan completos.

sin6_port: puerto TCP o UDP de 16 bits. Se almacena en network order.

sin6_addr: dirección IPv6 de 128 bits. Se almacena en network order.

sin6_scope_id: campo de 32 bits que se usa para identificar el alcance de la dirección llevada en el campo `sin6_addr`. Recordar que IPv6 define diferentes alcances: como Locales, de Sitio, Globales. El mapeo de estos valores a las interfaces particulares depende de cada implementación.

En los sistemas compatibles con 4.4BSD agregan un campo de 8 bits para indicar el tamaño en bytes de la estructura, `sin6_len` y se cambia la definición del tamaño para tipo `sa_family_t` que pasa de 16 a 8 bits, para dejar lugar al campo adicionado.

Listado 7.6: Socket address struct para BSD4.4

```
struct sockaddr_in6 {
    uint8_t         sin6_len;       /* length of this struct */
    sa_family_t     sin6_family;    /* AF_INET6 8bits*/
    in_port_t       sin6_port;      /* transport layer port # */
    uint32_t        sin6_flowinfo;  /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id;  /* set of interfaces
                                       for a scope */
};
```

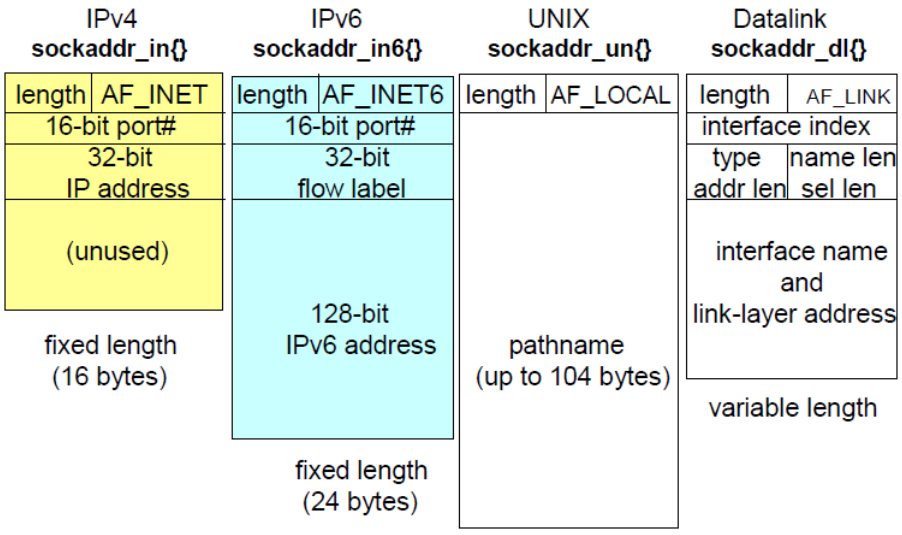


Figura 7.1: Varias estructuras de socket address

La estructura genérica, opaca, sobre la cual se monta (overlay) la estructura específica del protocolo es la siguiente.

Listado 7.7: Socket address struct Genérica

```
struct sockaddr {              /* only used to cast pointers */
    uint8_t      sa_len;
    sa_family_t  sa_family;    /* address family: AF_xxx value */
    char         sa_data[14]; /* protocol-specific address */
};
```

En la figura 7.1, de acuerdo al libro, UNIX Network Programming V1 [SFR03], se muestra una comparación entre diferentes estructuras de socket address que se mapean a la estructura genérica.

7.2.4. Draft de Socket API para asignar el Flow Label

Debido a que la RFC oficial no define nada sobre la programación utilizando el campo se ha propuesto un draft entre 2000 y 2001: "Socket API for IPv6 flow label field (draft-itojun-ipv6-flowlabel-api-01)" [Hag01] el cual intenta definir como sería su uso. El mismo nunca llegó a entrar en el circuito de los estándares. El documento indica que el Flow Label no debe ser manipulado por el usuario y que el kernel debe manejarlo para asegurar las condiciones estrictas definidas por la RFC. El kernel será el responsable de seleccionar un valor (pseudo)-aleatorio y único para el campo y mantenerlo durante la duración del flujo. Se muestra un código de ejemplo y referencia de como sería para consultar el valor que el kernel está usando.

Listado 7.8: Cómo obtener el flowlabel asignado por el kernel

```
struct sockaddr_in6 src, dst, altdst;
u_int32_t value;  /* the value for flow label */
int s;            /* socket */
socklen_t slen;

slen = sizeof(dst);
dst.sin6_flowinfo = 0;  /* must be zero */
connect(s, (struct sockaddr *)&dst, slen);

/* sent with the flow label field filled */
send(s, buf, buflen);

/* obtain the flow label value */
slen = sizeof(src);
getsockname(s, (struct sockaddr *)&src, &slen);
printf("flowlabel=%x\n", ntohl(src.sin6_flowinfo &
                               IPV6_FLOWLABEL_MASK));
```

También propone el uso de la función `setsockopt()` para indicar si el kernel debe asignarlos de forma automática o dejarlos en cero. Previo a usar `setsockopt()` se recomienda llamar a `getsockopt()` para obtener el valor que tenía seteado.

Listado 7.9: Propuesta de opción para el kernel según RFC-3493

```
const int off = 0;
const int on = 1;
int s;  /* socket */

/* disables automatic flow label */
setsockopt(s, IPPROTO_IPV6, IPV6_AUTOFLOWLABEL, &off,
           sizeof(off));
/* enables automatic flow label */
setsockopt(s, IPPROTO_IPV6, IPV6_AUTOFLOWLABEL, &on,
           sizeof(on));
```

7.3. Análisis de soporte en Sistemas Operativos

Para hacer un análisis del API ofrecida por sistemas operativos de código abierto como lo son GNU/Linux o los basados en BSD como OpenBSD o NetBSD se buscó documentación que explicase cómo es la programación para configurar el campo **Flow Label**. Para ninguno de los sistemas mencionados se encontró de forma fácil información que explicase su funcionamiento. Tampoco se encontraron gran variedad

de aplicaciones que lo utilizasen. Buscando de forma rápida lo mismo para sistemas propietarios como MS Windows o MacOS tampoco se encontró nada.

Como aplicación multi-plataforma se encontró `iperf3`. Esta es una re-escritura de la herramienta `iperf` desarrollada por NLANR/DAST y usada para poder medir de forma activa el rendimiento de la red. La idea de `iperf3` es desarrollar una herramienta desde cero donde el código sea más simple y tenga la capacidad de mantener una librería que puede ser usada por otros programas. La herramienta es soportada principalmente en los sistemas CentOS Linux, FreeBSD y MacOS X. Ha sido reportado su uso en las plataformas OpenBSD, Android y otras distribuciones de GNU/Linux. En el trabajo se la prueba sobre Debian GNU/Linux. El desarrollo de la herramienta es llevado a cabo mayormente por ESnet/Lawrence Berkleley National Laboratory `http://fasterdata.es.net` y se distribuye bajo un licenciamiento tipo BSD.

Para `iperf3` solo se encuentra disponible la opción de uso de **Flow Label** en la plataforma GNU/Linux y no compila o directamente no se setea en otros sistemas. Su funcionalidad es proveída mediante un patch que ya es parte del código. En su configuración, previo a la compilación, se puede observar que el soporte detectado parece solo estar presente para GNU/Linux.

```
$ cat source/iperf3/configure
...
# Check for IPv6 flowlabel support (believed to be Linux only)
# We check for IPV6_FLOWLABEL_MGR in <linux/in6.h> even though we
# don't use that file directly (we have our own stripped-down
# copy, see src/flowlabel.h for more details).
...
```

En el archivo `configure` se busca una macro `IPV6_FLOWLABEL_MGR` que solo se encuentra definida en los sistemas que tienen el kernel de Linux. La salida del comando muestra que la opción `-L` solo está disponible en la plataforma.

```
iperf3
 iperf3: parameter error - must either be a client (-c) or server (-s)

Usage: iperf [-s|-c host] [options]
       iperf [-h|--help] [-v|--version]
 ...
Client specific:
  -c, --client  <host>   run in client mode, connecting to <host>
  ...
  -4, --version4         only use IPv4
  -6, --version6         only use IPv6
  -S, --tos N            set the IP 'type of service'
  -L, --flowlabel N       set the IPv6 flow label (only supported on Linux)
  -Z, --zerocopy         use a 'zero copy' method of sending data
  -O, --omit N           omit the first n seconds
  -T, --title str        prefix every output line with this string
 ...
```

Otra herramienta que se encontró con soporte de manejo de Flow Label en GNU/-Linux sobre la plataforma Debian es `ping6(3)` incluida en el paquete `iputils-ping`, pero, por default, si se instala el paquete ".deb" esta opción viene deshabilitada.

```
$ ping6
Usage: ping6 [-LUdfnqrvVaAD] [-c count] [-i interval] [-w deadline]
             [-p pattern] [-s packetsize] [-t ttl] [-I interface]
             [-M pmtudisc-hint] [-S sndbuf] [-F flowlabel] [-Q tclass]
             [[-N nodeinfo-option] ...]
             [hop1 ...] destination

$ ping6 -F 0x03 ::1
Flow labels are not supported.
```

Para poder compilarlo con el soporte se requieren hacer algunos cambios en el código fuente.

```
$ diff iputils-20101006/Makefile.orig iputils-20101006/Makefile
4c4
< DEFINES=
---
> DEFINES=-DIPV6_FLOWLABEL_MGR

$ diff iputils-20101006/ping6.c.
orig iputils-20101006/ping6.c
70c70
<
---
> #include "flowlabel.h"
```

Además se debe generar el C header `flowlabel.h`, ya que los paquetes de desarrollos no incluyen ninguno un encabezado para las componentes necesarias para compilarlo. Si bien GNU/Linux provee encabezados para el Flow Label, estos son internos, y al compilar, generan conflictos. Las estructuras están provistas en el header `linux/in6.h` pero si se incluye se genera conflicto con re-definiciones encontradas en el header público `netinet/in.h`. Otra característica que tiene la herramienta es requerir el privilegio de SUID (set-uid) debido a que gran parte está construida en socket RAW. Incluso esta cuestión en los comentarios del código fuente se menciona como bug.

```
$ cat iputils-20101006/ping6.c.orig
 ...
 * Bugs -
 *      More statistics could always be gathered.
 *      This program has to run SUID to ROOT to access the ICMP socket.
 ...
```

Con respecto a los sistemas BSD, en el artículo [MM05] se hace un testeo de varios sistemas operativos y su uso del Flow Label. En el mismo se indica que se prueba FreeBSD, que como todos los sistemas BSD usan de base la implementación del stack IPv6 del proyecto KAME. La implementación según los autores, se supone

que para las conexiones TCP hace el uso del campo de acuerdo a [Hag01] colocando un valor (pseudo)-aleatorio, aunque se encontró que lo dejaba en cero. Para arreglar el problema se envió por parte de los autores del artículo [MM05] un pequeño patch que consistía en el código mostrado, donde se indica que se genere un valor por el kernel.

Listado 7.10: patch para FreeBSD de David Malone

```
/* update flowinfo - draft-itojun-ipv6-flowlabel-api-00 */
 inp->in6p_flowinfo &= ~IPV6_FLOWLABEL_MASK;
 if (inp->in6p_flags & IN6P_AUTOFLOWLABEL)
 inp->in6p_flowinfo |= (htonl(ip6_randomflowlabel()) &
                        IPV6_FLOWLABEL_MASK);
```

Para UDP se ve que va siempre en cero. Los sistemas BSD basados en KAME Project, a partir de las versiones FreeBSD 4.0, OpenBSD 2.7, NetBSD 1.5. parecen tomar la postura de no ofrecer al usuario una API para setear el valor del campo directamente. Los sistemas GNU/Linux basados en la implementación de USAGI (UniverSAl playGround for Ipv6) Project parece ser más flexible en este aspecto.

7.3.1. Soporte de Flow Label en GNU/Linux

Para GNU/Linux la estructura de dirección es similar a la definida por el RFC-3493 `ipv6(7)`. Solo cambia el nombre de los tipos de los dos primeros campos.

Listado 7.11: Socket address struct IPv6 en GNU/Linux

```
struct sockaddr_in6 {
    u_int16_t       sin6_family;   /* AF_INET6 */
    u_int16_t       sin6_port;     /* port number */
    u_int32_t       sin6_flowinfo; /* IPv6 flow information */
    struct in6_addr sin6_addr;     /* IPv6 address */
    u_int32_t    sin6_scope_id;  /* Scope ID (new in 2.4) */
};

struct in6_addr {
    unsigned char           s6_addr[16];/* IPv6 address */
};
```

Los campos tienen los mismos significados ya analizados. `sin6_family` siempre seteado al valor `AF_INET6`. `sin6_flowinfo` es el flow ID de IPv6 y puede contener también el **Traffic Class**. `sin6_scope_id` es el ID del alcance de la dirección en la interfaz. Por ahora GNU/Linux, a partir de kernel 2.4, solo lo soporta para direcciones de alcance de link.

Como documentación para uso del Flow Label en GNU/Linux solo se encontró el código fuente de iperf3 y un página web en la URL:
`http://xpg.dk/projects/linux-ipv6-flow-label-mangement-api/` del 2013 donde se menciona que no se ha podido tampoco encontrar más referencias del tema. En GNU/Linux para poder setear el **Flow Label** en el encabezado IPv6 se requiere primero reservar el valor mediante el kernel. El kernel administra los valores de forma de adherir a las especificaciones de la RFC de manera que sean únicos en un período de tiempo. Para poder generar código de usuario, como se comentó con `ping6(3)`, se requieren generar headers a nivel de usuario con las constantes, macros y estructuras necesarias para compilar el código. Estos valores deben ser los mismos que los que se proveen internamente, por lo que copiando parcialmente los datos necesarios a un nuevo header es suficiente.
El kernel de Linux requiere recibir el requerimiento de reserva del recurso desde la aplicación usando la función `setsockopt()` pasándolo mediante una estructura de datos definida de la siguiente forma:

Listado 7.12: Estructura para comunicar opciones de flowlabel en Linux

```
struct in6_flowlabel_req {
    struct in6_addr flr_dst;
    __u32 flr_label;
    __u8 flr_action;
    __u8 flr_share;
    __u16 flr_flags;
    __u16 flr_expires;
    __u16 flr_linger;
    __u32 __flr_pad;
};
```

A través de la misma se puede solicitar los recursos, liberarlos o renovarlos. El significado de los campos se describe a continuación:

flr_dst: la dirección IPv6 destino para el flow.

flr_label: el valor de 0 bit del flowID en network orden.

flr_action: la acción solicitada:

IPV6_FL_A_GET (0): para obtener un **Flow Label**. Si el valor del flowID no es cero y se usa el flag de uso exclusivo se obtiene este ID con este modo de uso. Si el valor es cero y esta presente la opción `IPV6_F_F_CREATE` se obtiene un nuevo valor para el flowID asignado por el kernel, el código obtenido es retornado en el campo `flr_label`.

IPV6_FL_A_PUT (1): libera el **Flow Label** indicado.

IPV6_FL_A_RENEW (2): renueva el tiempo de uso del recurso.

flr_share: como se comporte dentro del sistema el valor del **Flow Label**.

IPV6_FL_S_NONE (0): no se puede compartir en ningún momento.

IPV6_FL_S_EXCL (1): uso exclusivo por parte del socket que hace la solicitud. se puede usar luego de liberarse.

IPV6_FL_S_PROCESS (2): compartido por los sockets del mismo proceso.

IPV6_FL_S_USER (3): compartido por los sockets del mismo usuario.

IPV6_FL_S_ANY (255): permite compartir en cualquier condición.

flr_flags: flags posibles. IPV6_FL_F_CREATE (1) y IPV6_FL_F_EXCL (2). El primero se debe usar para solicitar un nuevo valor generado por el kernel si el campo flr_label es cero y se hace un GET.

flr_expires: parece ser el tiempo de expiración. No se tiene definida la unidad.

flr_linger: parece ser un indicador, si se permite persistencia o no del flowID.

Todo está definido en el kernel a partir de las versiones 2.6, al menos, y se encuentra también en la serie 3.0 en el header interno. La opción para usar la administración de **Flow Labels** por el kernel es `IPV6_FLOWLABEL_MGR (32)` y su forma de uso es la siguiente.

Listado 7.13: Uso de la opción de **Flow Label** manager

```
struct in6_flowlabel_req *freq =
       (struct in6_flowlabel_req *)freq_buf;
int freq_len = sizeof(*freq);
...
setsockopt(sock, IPPROTO_IPV6, IPV6_FLOWLABEL_MGR, freq,
                 freq_len)
```

Todos los paquetes pertenecientes al mismo flujo deben compartir todas las opciones. Una vez solicitado y obtenido el valor desde el kernel es necesario indicar que se debe habilitar el valor en los envíos de los paquetes. Esto se realiza usando otra opción binaria: `IPV6_FLOWINFO_SEND (33)`.

Listado 7.14: Uso de la opción de **Flow Label** send

```
int on = 1;
...
setsockopt(sock, IPPROTO_IPV6, IPV6_FLOWINFO_SEND, &on,
                sizeof(on))
```

Los datos en el header interno son los siguientes:

Listado 7.15: Contenido parcial del header interno que incluye opciones de flow label

```
$ cat include/linux/in6.h
...
struct in6_flowlabel_req {
        struct in6_addr flr_dst;
        __be32  flr_label;
        __u8    flr_action;
        __u8    flr_share;
        __u16   flr_flags;
        __u16   flr_expires;
        __u16   flr_linger;
        __u32   __flr_pad;
        /* Options in format of IPV6_PKTOPTIONS */
};

#define IPV6_FL_A_GET    0
#define IPV6_FL_A_PUT    1
#define IPV6_FL_A_RENEW  2

#define IPV6_FL_F_CREATE        1
#define IPV6_FL_F_EXCL          2

#define IPV6_FL_S_NONE          0
#define IPV6_FL_S_EXCL          1
#define IPV6_FL_S_PROCESS       2
#define IPV6_FL_S_USER          3
#define IPV6_FL_S_ANY           255
...
#define IPV6_FLOWINFO           11
...
/* Flowlabel */
#define IPV6_FLOWLABEL_MGR      32
#define IPV6_FLOWINFO_SEND      33
```

7.3.2. Manejo de congestión por TCP en GNU/Linux

Para las pruebas realizadas se requieren herramientas que permita testear la red y analizar el comportamiento del protocolo. Una posibilidad es utilizar directamente el paquete iperf3, pero como parte del desarrollo de la tesis es conveniente aplicar los conocimientos obtenidos con el Flow Label en IPv6 en la programación de una herramienta de testeo. La utilidad aplica las funcionalidades ofrecidas por el sistema operativo GNU/Linux con respecto al marcado de los Flow Label y la posibilidad de seleccionar el algoritmo para TCP de control de congestión [MBM13], abreviado CC. El algoritmo de control de congestión se utiliza para mostrar el desempeño de estos en competencia.

Para la selección del algoritmo de control de congestión en GNU/Linux para TCP se lo puede hacer usando el API que ofrece el kernel mediante el file system "/proc" o mediante la utilidad `sysctl(8)`. Por ejemplo para consultar al algoritmo activo se pueden ejecutar los comandos mostrados a continuación.

```
# cat /proc/sys/net/ipv4/tcp_congestion_control
cubic

# sysctl net.ipv4.tcp_congestion_control
net.ipv4.tcp_congestion_control = cubic
```

En la nomenclatura del kernel sigue figurando el protocolo IPv4, pero el cambio también se aplica IPv6 debido a que es sobre TCP y no la capa de red. Para modificarlo también se puede realizar desde la misma interfaz que ofrece el sistema.

```
# sysctl net.ipv4.tcp_congestion_control=vegas
net.ipv4.tcp_congestion_control = vegas

# sysctl net.ipv4.tcp_congestion_control=reno
net.ipv4.tcp_congestion_control = reno

# sysctl net.ipv4.tcp_congestion_control=htcp
net.ipv4.tcp_congestion_control = htcp
```

En este caso se puede ver que, tanto Reno como Cubic (la opción default en Linux) se implementan directamente en el kernel y no como módulos a diferencia de los otros dos ejemplos, Vegas y HTCP.

```
# lsmod | grep "tcp_"
tcp_htcp               13041  0
tcp_vegas              13891  0
```

En este texto solo se mostraron algunas variantes, de todas las disponibles en GNU/Linux. En los artículos: [SK02], [MN06], [Pfe07], y en la URL: `https://fasterdata.`

es.net/host-tuning/linux se puede ver el tema en más detalle. De forma análoga se puede indicar desde la opciones del socket el algoritmo a utilizar mediante la llamada a `setsockopt()`.

Listado 7.16: Uso de la opción de Algoritmo de CC en TCP

```
char cgctrl[MAX_NLEN];
int optlen;
...
strncpy(cgctrl, "reno", MAX_NLEN);
optlen = strnlen(cgctrl,MAX_NLEN);
setsockopt(sock, IPPROTO_TCP, TCP_CONGESTION,
           (char*) cgctrl, (size_t) optlen);
```

7.4. Herramientas desarrolladas

7.4.1. Herramienta de marcado de FL

A partir del API y las facilidades ofrecidas por el sistema operativo se programaron las herramientas para el testing. Se desarrolló un programa para generar y leer tráfico TCP. El programa tiene una gran cantidad de parámetros para poder hacer testing.

```
# ./tcp6-socky
usage: ./tcp6-socky -C|-L
        [-l|--local <src-ip>] [-p|--port <port>]
        [-w|--wait <sec>] [-R|--recvbuf <size>] [-S|--sendbuf <size>]
        [-m|--mss <size>] [-r|--reuse]  [-n|--nodelay] [-k|--keepalive]
        [--help|-h]
  For -L: [-b|--blog <num>] [-f|--fork|-t|--thread|-u|--unique]
  For -C: -d|--dst <dst-ip> [-z|--sport <port>] [-X|--flabel <label>
                            [-G|--cgctrl reno|vegas|cubic|...]
  -L stands for Listen (Server Mode), -C stands for Client
For both, extra options
         [-E|--extra <arg1> <arg2> ... <argn>]
Extra args may be: +                        (interactive)
                   h [times] [delay]        (hole by char)
                   k [times] [chunk]        (hole by chunk)
                   e [times] [delay] [tmout] (echo chars and delay or timeout)
                   p [times] [delay] [tmout] (print chars and delay or timeout)
                   C [times] [delay] [tmout] (gen seq chars delay and tmout)
                   R [times] [delay]        (gen random chars)
                   A [times] [delay] [rchar] (rep chars delay)
                   K [times] [chunk] [tmout] (gen chars by chunk)
                   T [times] [delay]        (gen time)
                   times = 0 means infinity
                   delay = 0 means no delay
                   tmout = 0 means no timeout
```

También un programa para generar y leer tráfico UDP.

```
# ./udp6-socky
usage: ./udp6-socky  -R|--recv || -R|--recv -y|--synchar
           -S|--sendto <dst-ip> |||
           -S|-sendto <dst-ip> || -y|--synchar
         [-F|--from <src-ip>] [-z|--sport <port>] [-p|--port  <port>]
         [-l|--size <size>] [-c|--count <count>]  [-X|--flabel <label>
```

```
[-d|--delay <usec>] [-r|--reuse] [-C|--connect]
[-m|--mcast] [-n|--mtudisc]  [-r|--recvbuf <size>]
[-s|--sendbuf <size>]
[--help|-h]
```

Por ejemplo para generar tráfico con el flowlabel con el valor **60** en decimal o **0x3C** se puede realizar de la siguiente manera. En el cliente TCP la opción `-C` marca el modo cliente, `-r` marca la opción de reuse addr para evitar el 2MSL en el caso de cierre abrupto del servidor, `-X` indica el valor del campo Flow Label, `-d` indica el destino y `-E` indica el parámetro extra, en este caso `K` para que genere datos en "chunks" (porciones) por 10 segundos.

```
root@n5:/tmp# ./tcp6-socky -C -r -X 60 -d 2001:db8:4::1 -E K 10
RBUF:      87380
SBUF:      16384
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Trying to connect to 2001:db8:4::1:8000
Connected to 2001:db8:4::1:8000
RBUF:      87380
SBUF:      21280
MSS:       1428
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Extra Options: opt=K times=10 delay=0 chunk=8192 rchar=A tmout=0
proc 10 bytes in 1983.000000 usec or 0.001983 sec
BW=0.040343 Mbps
 Return: 0
```

y corriendo en el servidor, modo server `-L` listen, con la opción extra `-E` de `k` para consumir los datos recibidos descartándolos.

```
root@n4:/tmp# ./tcp6-socky -L -r -E k
RBUF:      87380
SBUF:      16384
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Waiting connection on 0:::8000 backlog: 1 model: 0
Connected from 2001:db8:20::100:58895
Extra Options: opt=k times=0 delay=0 chunk=8192 rchar=A tmout=0

proc 10 bytes in 3425.000000 usec or 0.003425 sec
BW=0.023358 Mbps
 Return: 0
Waiting connection on 0:::8000 backlog: 1 model: 0
```

En la figura 7.2 se ve la captura del paquete IPv6 que contiene un segmento TCP generado con las pruebas anteriores. Se puede observar la marca del campo Flow Label.

De forma similar, se pueden marcar los datagramas UDP con la otra herramienta desarrollada, con el valor decimal **8888** o hexadecimal **0x22B8**. El paquete se muestra en la figura 7.3. En ese caso se usan las opciones `-S` (send), `-l` para el tamaño del datagrama y `-c` para solo generar 10 (diez) de los mismos, pues solo se quiere ver el marcado.

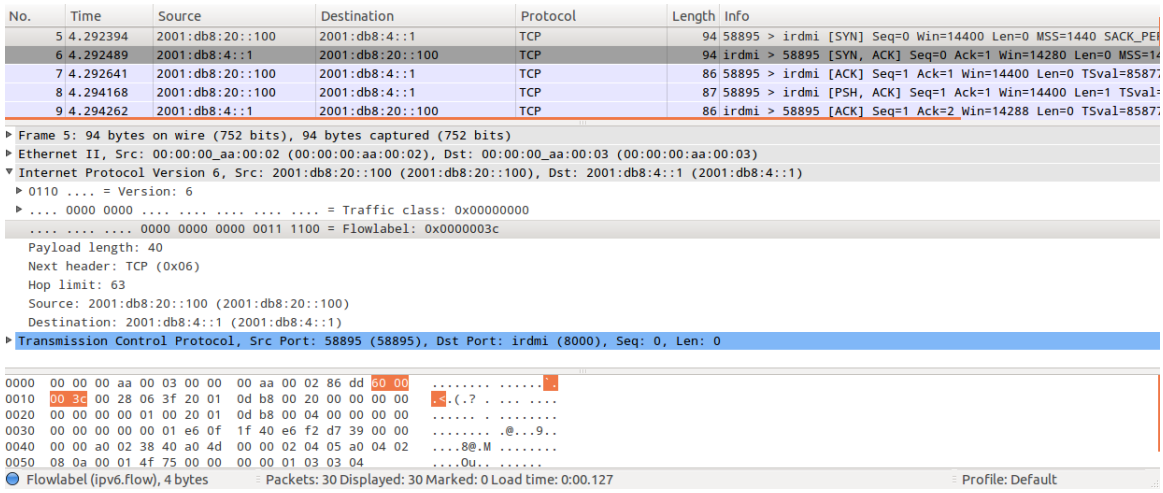


Figura 7.2: Paquete IPv6 con el FL marcado llevando segmento TCP

```
root@n5:/tmp# ./udp6-socky -S 2001:db8:4::1 -X 8888 -l 1000 -c 10
RBUF:       163840
SBUF:       163840
MCAST:      0
REUSE:      0
MTUDISC:    1
```

El código de referencia de testing de las herramientas desarrolladas se encuentra en la URL: `https://gitlab.com/andres.barbieri/ipv6-socky` con acceso abierto.

7.4.2. Herramienta de comparación de FL

Como herramientas de matching, comparación del campo, la cantidad de opciones tampoco es muy extensa en los sistemas operativos. Se ha encontrado el soporte de las herramientas de GNU/Linux: ip6tables y TC mediante los módulos de u32, que permiten comparar cualquier porción del paquete de a 32bits. El módulo `u32` permite testear haciendo matching de valores tomados de a 4 bytes desde el paquete IPv6. La flexibilidad que ofrece es suficiente como para encontrar cualquier valor dentro de los campos del encabezado del paquete y de su contenido, por ejemplo el encabezado TCP.

El formato de la expresión a comparar usa alguno de los siguientes operadores `"&"`, `"<<"`, `">>"` , que tienen los significados similares que en el lenguaje Ç”. Otros operadores que encontramos son `=` y `"&&"`. Las expresiones más comunes tienen la forma mostrada a continuación.

```
"Offset&Mask=Value"
```

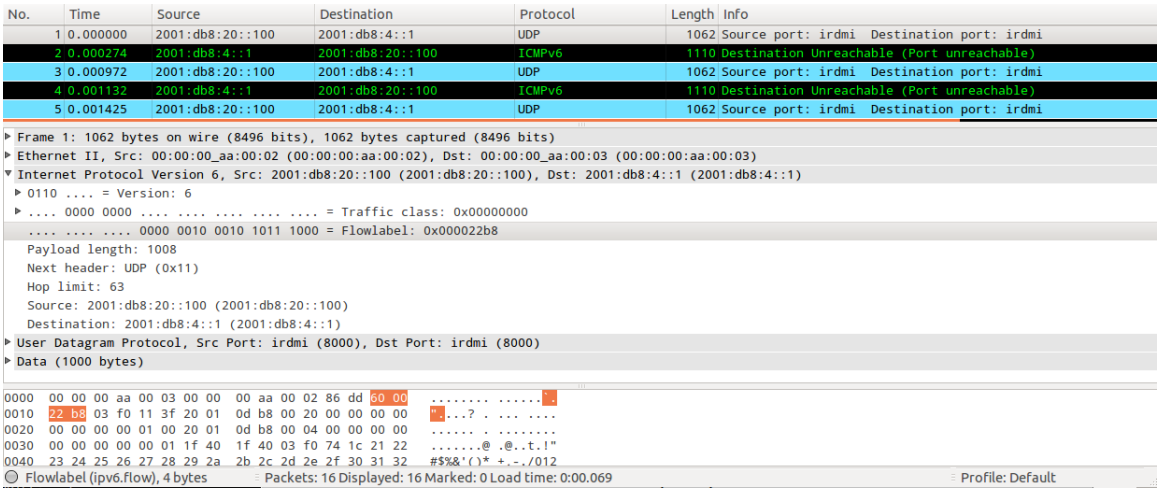


Figura 7.3: Paquete IPv6 con el FL marcado llevando datagrama UDP

`"Offset&Mask=First:Last"`

El primer valor de la expresión en general es el offset, desplazamiento del principio del encabezado del paquete IPv6. Seguido habitualmente se encuentra el operador `"&"` que indica que se haga el matching con un "AND" lógico a partir del offset contra la máscara dada y comparar el resultado contra el valor o que este pertenezca al rango "First..Last". Mirando los primeros 32 bits del paquete IPv6 de acuerdo al diagrama siguiente.

```
  0123 45678901 23456789012345678901   Total bits
 +----+--------+----------------------+
 |Ver | TCLass | Flow label           |
 +----+--------+----------------------+
  0123 01234567 01234567890123456789   Field bits

  0000 00000000 11111111111111111111   MASK
```

En el caso de comparar el Flow Label las siguientes expresiones son ejemplos. Primero una expresión que compara todos los valores posibles de FL.

```
--u32 "0&0x000FFFFF=0x0:0xFFFFF"
```

Una expresión que compara el valor particular de Flow Label que sea igual a 0x3C.

```
--u32 "0&0x000FFFFF=0x0003C:0x0003C"
--u32 "0&0x000FFFFF=0x0003C"
```

El valor particular de Flow Label se encuentra entre 0x00010 y 0x0ABCD.

```
--u32 "0&0x000FFFFF=0x00010:0x0ABCD"
```

Se pueden aplicar estas expresiones directamente a ip6tables logrando "logear" (enviar a los logs del sistema) acorde pasa el tráfico y ver la cantidad de mensajes marcados con determinado valor. A continuación se muestra un ejemplo de su uso.

```
root@n1#
 ip6tables -I FORWARD -m u32 --u32 "0&0x000FFFFF=0x0003C" -j LOG

root@n1#
 ip6tables -I FORWARD -m u32 --u32 "0&0x000FFFFF=0x02200:0x0FFFF"
           -j LOG --log-prefix UDP-mrk

root@n1# ip6tables -L -n -v | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source       destination
    0     0 LOG        all      *      *       ::/0                ::/0
    u32 "0x0&0xfffff=0x2200:0xffff" LOG flags 0 level 4 prefix "UDP-mrk"
    0     0 LOG        all      *      *       ::/0                ::/0
    u32 "0x0&0xfffff=0x3c" LOG flags 0 level 4
```

Luego generar los paquetes y ver como se incrementan los contadores.

```
root@n5# ./tcp6-socky -C -r -X 60 -d 2001:db8:4::1 -E K 10

root@n1# ip6tables -L -n -v | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source       destination
    0     0 LOG        all      *      *       ::/0                ::/0
    u32 "0x0&0xfffff=0x2200:0xffff" LOG flags 0 level 4 prefix "UDP-mrk"
  14  1026 LOG        all      *      *       ::/0             ::/0
    u32 "0x0&0xfffff=0x3c" LOG flags 0 level 4
```

De forma similar para UDP también se muestra como se incrementan los contadores para la regla de matching.

```
root@n5# ./udp6-socky -S 2001:db8:4::1 -X 8888 -l 1000 -c 10

root@n1# ip6tables -L -n -v | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 42 packets, 18954 bytes)
 pkts bytes target     prot opt in     out     source         destination
  10 10480 LOG        all      *      *       ::/0              ::/0
 u32 "0x0&0xfffff=0x2200:0xffff" LOG flags 0 level 4 prefix "UDP-mrk"
   14  1026 LOG        all      *      *       ::/0                   ::/0
 u32 "0x0&0xfffff=0x3c" LOG flags 0 level 4
```

Se pueden ver los logs generados en los archivos del host que hace el procesamiento.

```
root@n1# tail /var/log/syslog
...
Jul 12 16:31:40 vcore kernel: [ 2897.382828] IN=eth0 OUT=eth1
    MAC=00:00:00:aa:00:03:00:00:00:aa:00:02:86:dd
    SRC=2001:0db8:0020:0000:0000:0000:0000:0100
    DST=2001:0db8:0004:0000:0000:0000:0000:0001 LEN=80
    TC=0 HOPLIMIT=62 FLOWLBL=60 PROTO=TCP SPT=33886 DPT=8000
    WINDOW=14400 RES=0x00 SYN URGP=0
...
Jul 12 16:32:25 vcore kernel: [ 2941.602491] UDP-mrkIN=eth0 OUT=eth1
    MAC=00:00:00:aa:00:03:00:00:00:aa:00:02:86:dd
    SRC=2001:0db8:0020:0000:0000:0000:0000:0100
    DST=2001:0db8:0004:0000:0000:0000:0000:0001 LEN=1048
    TC=0 HOPLIMIT=62 FLOWLBL=8888 PROTO=UDP SPT=8000 DPT=8000 LEN=1008
```

Con esto se tiene el conjunto de herramientas necesarias para marcar el campo y dar un tratamiento diferenciado de QoS. Se puede acordar que en el campo Flow Label estará marcado el tratamiento que el paquete, o mejor indicado el flujo, debe recibir. De esta forma es posible especificar el tratamiento que recibirá el tráfico generado cuando pasa por el dominio de QoS. La primera idea a implementar es pre-acordar

el valor del ancho de banda que se debe reservar, marcarlo por la aplicación origen y, luego, con la herramienta TC implementar las restricciones para implementarla. El desarrollo y las pruebas se muestran en el siguiente capítulo.

Capítulo 8

Implementación usando el Flow Label para QoS

8.1. Primer propuesta del uso del campo Flow Label para marcar QoS en GNU/Linux

La propuesta es implementar un modelo en el cual el valor del campo Flow Label indique el tratamiento particular de la QoS. El origen marcará en este un valor previamente acordado con la red, de forma similar como sucede con DiffServ. Este valor indicará la QoS necesaria. Cuando los mensajes lleguen a los nodos de la red estos sabrán interpretar el valor que lleva y lo manejarán de acuerdo a lo acordado. En los ejemplo presentados en el documento no se usa un protocolo de señalización, sino se acuerdan los valores y el tratamiento mediante una configuración manual estática. El modelo no prohíbe el uso de un protocolo de señalización, pero agregar uno sería bastante complejo y se perdería el objetivo de lo que se quiere mostrar que es el tratamiento diferenciado. En el primer enfoque se usa el campo para codificar un valor donde se indique el ancho de banda digital requerido.

8.1.1. Tests realizados, primer propuesta

Para probar la utilidad del campo Flow Label en un tratamiento diferenciado de QoS se montó una maqueta de equipos como la mostrada en la figura 8.1. La misma se replicó en equipos físicos, 2 núcleos AMD y 3GB de RAM, doble NIC Ethernet 1000BaseTX ejecutando el SO GNU/Linux y en equipos virtuales, ejecutando Linux Containers (LXC).

En la maqueta se restringe el ancho de banda del enlace central a 10Mbps, dejando el resto sin cambios. El objetivo de este cambio es para ver el comportamiento del

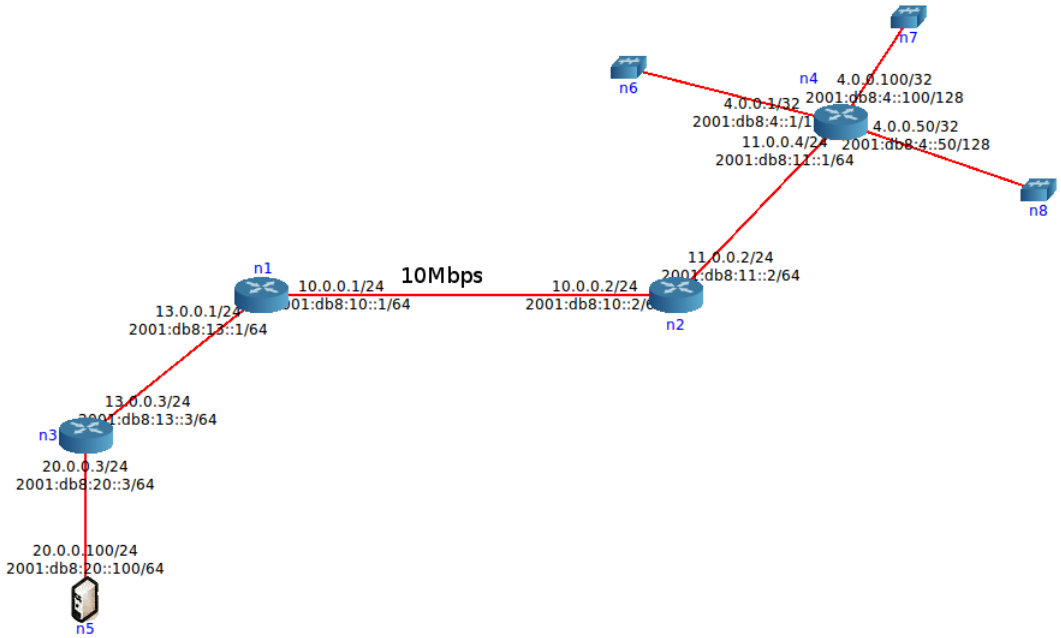


Figura 8.1: Topología de testing

tráfico cuando los recursos son limitados, propósito que persigue la QoS. Para las pruebas se genera tráfico desde el nodo **n5** hacia las direcciones IP en el otro extremo. Todas las pruebas con TCP, que no se indique lo contrario, se hacen con el algoritmo de control de congestión (CC) default, que para la versión de kernel usada es Cubic. En aquellas que se realizan los cambios explícitamente se indica que algoritmo de CC se usa.

8.1.1.1. Tests Individuales a 10Mbps

Previo al tratamiento diferenciado de la QoS de acuerdo al campo Flow Label se realizan las pruebas de forma individual, sin que el tráfico compita, con el enlace intermedio regulado a 10Mbps. Una forma de limitarlo es colocando las placa de red a 10Mbps FDX de forma forzada, por ejemplo, con las herramientas del sistema `ethtool(8)` `mii-tool(8)` si la NIC lo soporta.

```
root@n1:/tmp#  ethtool -s eth1 speed 10 duplex full [autoneg off]
```

Otra alternativa es limitarlo con la herramienta `tc-netem(8)` con el módulo `netem` de emulación de red: `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`.

```
root@n1:/tmp# ip6tables -t mangle -F
root@n1:/tmp# ip6tables -t filter -F
root@n1:/tmp# tc qdisc add dev eth1 root tbf rate 10mbit latency 0.5ms burst 20KB
```

A continuación se muestran los comandos para generar las pruebas y capturar el tráfico para analizarlo más tarde. En el primer cuadro de comandos se captura y se genera el tráfico UDP con un delay de `-d` 1500 microsegundo entre datagramas, donde el tamaño es de `-l` 1400 bytes y genera, `-c` 4500. Se marca con el valor de Label 10 expresado en base decimal.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w fl-udp.pcap

root@n5:/tmp# time ./udp6-socky -S 2001:db8:4::50 -d 1500 -X 10 -l 1400
              -c 4500
RBUF:      163840
SBUF:      163840
MCAST:     0
REUSE:     0
MTUDISC:   1

real 0m10.811s
user 0m0.008s
sys 0m2.664s
```

En el escenario presentado a continuación se generan segmentos TCP durante 13 segundos con la marca de FL 5000 destinados al port 8001.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w fl-tcp-8001.pcap

root@n5:/tmp# ./tcp6-socky -C -r -X 5000 -d 2001:db8:4::100
              -p 8001 -E K 0 1400 13
RBUF:      87380
SBUF:      16384
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Trying to connect to 2001:db8:4::100:8001
Connected to 2001:db8:4::100:8001
RBUF:      87380
SBUF:      21280
MSS:       1428
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Extra Options: opt=K times=0 delay=0 chunk=1400 rchar=A tmout=13

proc 15334200 bytes in 13001476.000000 usec or 13.001476 sec
BW=9.435359 Mbps
 Return: 0

root@n4:/tmp# ./tcp6-socky -L -p 8001 -r -E k
```

En este otro caso se generan segmentos TCP durante 13 segundos con la marca 2000 al port 8002.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w fl-tcp-8002.pcap

root@n5:/tmp# ./tcp6-socky -C -r -X 2000 -d 2001:db8:4::100
              -p 8002 -E K 0 1400 13
RBUF:      87380
SBUF:      16384
```

```
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Trying to connect to 2001:db8:4::100:8002
Connected to 2001:db8:4::100:8002
RBUF:      87380
SBUF:      21280
MSS:       1428
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Extra Options: opt=K times=0 delay=0 chunk=1400 rchar=A tmout=13

proc 15310400 bytes in 13000334.000000 usec or 13.000334 sec
BW=9.421543 Mbps
 Return: 0

root@n4:/tmp# ./tcp6-socky -L -p 8002 -r -E k
...
```

El último test se realiza con `iperf(8)` sin marcar el tráfico.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w fl-tcp-5001-iperf.pcap

root@n5:/tmp# iperf -V -c 2001:db8:4::1
------------------------------------------------------------
Client connecting to 2001:db8:4::1, TCP port 5001
TCP window size: 20.8 KByte (default)
------------------------------------------------------------
[  3] local 2001:db8:20::100 port 41609 connected with 2001:db8:4::1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  11.4 MBytes  9.45 Mbits/sec

root@n4:/tmp# iperf -s -V
...
```

El marcado de los paquetes en todos los ejemplos no se tiene en cuenta debido a que no hay competencia, ya que cada prueba se realiza individualmente. Para los tests no se ha configurado ningún tratamiento del tráfico en los nodos de la red, el router intermedio no hace manejo especial en estas pruebas. Todas las evaluaciones TCP dan como resultado aprox. 10Mbps de utilización que es la limitación del enlace y la realizada sobre UDP da 5Mbps que es la limitación impuesta por el comando de acuerdo a la cantidad de tráfico que se quiere generar. Las gráficas de la pruebas TCP con la herramienta desarrollada, con iperf y con UDP se muestran en las figuras 8.2, 8.3 y 8.4 respectivamente. Las figuras muestran el eje de las abscisas sobre la derecha y en escala de 10bits de acuerdo a como se genera con el wireshark, por lo que 1.000.000 significa 10Mb (esto sucede con todas las gráficas en adelante).

Para la realización de las pruebas varias veces, es decir repetirlas, es recomendado configurar el salvado de métricas a deshabilitado mediante el siguiente comando.

```
root@n5:/tmp# sysctl -w net.ipv4.tcp_no_metrics_save=1
```

Habitualmente GNU/Linux "recuerda" el umbral de Slow Start: **slow start threshold (ssthresh)**, por lo tanto las pruebas sucesivas con TCP estarán influenciada por las anteriores. Debido a esto la recomendación de desactivar el salvado.

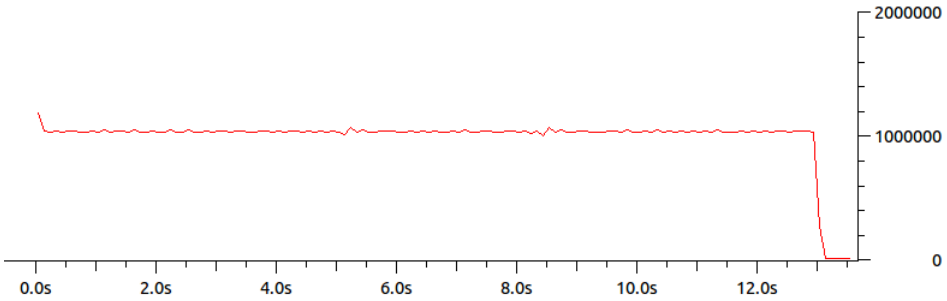


Figura 8.2: Test individual tcp6-socky al port 8002

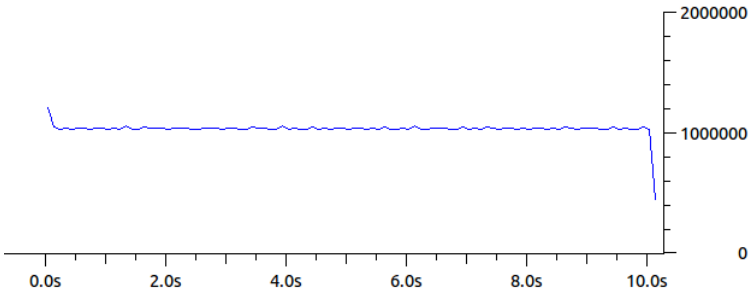


Figura 8.3: Test individual iperf al port 5001

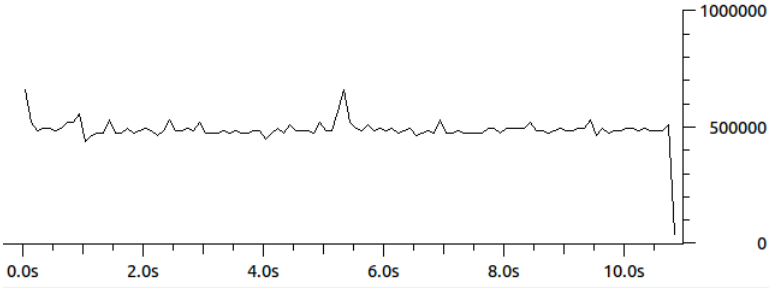


Figura 8.4: Test individual udp6-socky UDP

8.1.1.2. Tests combinados a 10Mbps sin QoS

Si se realizan los tests combinando los tráficos, compartiendo el enlace de 10Mbps sin configurar QoS todos competirán entre sí intentando usar la capacidad máxima pero obteniendo solo lo que el resto de los flujos le permitan.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w fl-combinadas.pcap

root@n5:/tmp# iperf -V -c 2001:db8:4::1 &
 ./tcp6-socky -C -r -X 2000 -d 2001:db8:4::100 -p 8002 -E K 0 1400 13 &
 ./tcp6-socky -C -r -X 5000 -d 2001:db8:4::100 -p 8001 -E K 0 1400 13 &
 time ./udp6-socky -S 2001:db8:4::50 -d 1500 -X 10 -l 1400 -c 4200

root@n4:/tmp# iperf -s -V &
              ./tcp6-socky -L -p 8001 -r -E k &
              ./tcp6-socky -L -p 8002 -r -E k

...
proc 4503380 bytes in 13042161.000000 usec or 13.042161 sec
BW=2.762352 Mbps
 Return: 0
Waiting connection on 0:::8001 backlog: 1 model: 0

...
proc 4188856 bytes in 13046443.000000 usec or 13.046443 sec
BW=2.568581 Mbps
 Return: 0
Waiting connection on 0:::8002 backlog: 1 model: 0

...
[  4] local 2001:db8:4::1 port 5001 connected with 2001:db8:20::100
      port 54063
[ ID] Interval       Transfer      Bandwidth
[  4]  0.0-10.5 sec  1.25 MBytes   998 Kbits/sec
```

En el gráfico 8.5 se ve la competencia de todos los tráficos por el ancho de banda total de 10Mbps, en el enlace más chico. Se observa en color negro el tráfico UDP que no se auto-regula por lo que se mantiene como CBR (constant bit rate) a 5Mbps. Luego los diferentes tráficos TCP se presentan con picos y valles debido al algoritmo de control de congestión del protocolo. Todos los flujos TCP compiten por los 5Mbps restantes que deja el tráfico UDP sin uso. La gráfica muestra que ninguno de los flujos TCP encuentra un buen aprovechamiento y tienen muchas subidas y bajadas. Hay que considerar que ese tráfico UDP que se mantiene constante, en el receptor no necesariamente mantiene la tasa de arribo y parte de sus datagramas se pierden.

En los últimos segundos de prueba, el tráfico UDP termina y se ve la competencia directa entre TCP. En la figura 8.6 se muestran los resultados generando tráfico todo con la herramienta `iperf` para comprobar que los resultados son similares y que no se deben a la introducción del programa desarrollado, en este caso se generan solo dos flujos TCP con Reno.

Para ver la pérdida UDP en el receptor, con la herramienta iperf se puede apreciar, aunque solo para IPv4, al menos con la versión utilizada para los tests: iperf version

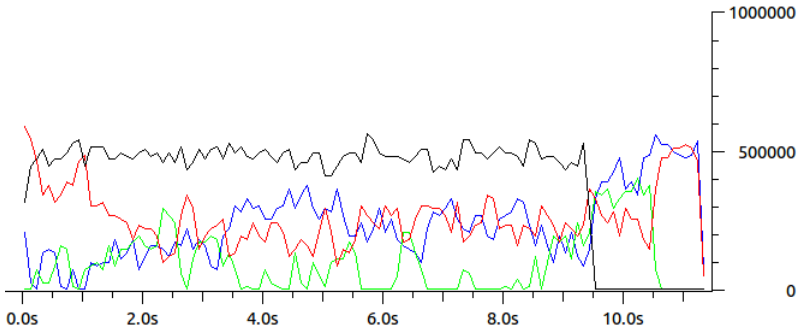


Figura 8.5: Test combinado sin aplicar QoS

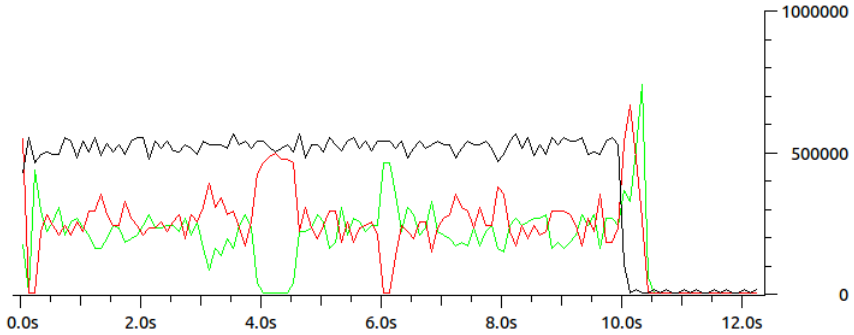


Figura 8.6: Test combinado sin aplicar QoS con iperf

2.0.5 (08 Jul 2010) pthreads. Para IPv6 no reporta estas variables. En el caso de que UDP compita contra otro tráfico también tendrá un porcentaje de pérdida como se muestra en la siguiente salida donde se general el tráfico UDP con IPv4.

```
root@n5:/tmp# iperf -c 2001:db8:4::1 -p 5001 -V &
    iperf -c 2001:db8:4::1 -p 5002 -V &
    iperf -c 11.0.0.4 -u -b 5Mbps
...
[ 3] local 2001:db8:20::100 port 57280 connected with 2001:db8:4::1
     port 5002
[ID] Interval       Transfer     Bandwidth
[ 3]  0.0-10.0 sec  6.25 MBytes  5.24 Mbits/sec
[ 3] Sent 4460 datagrams
[ 3] Server Report:
[ 3] 0.0-10.0 sec 5.83 MBytes 4.89 Mbits/sec 1.360ms297/4459(6.7%)
[ 3] 0.0-10.0 sec 9 datagrams received out-of-order

[ 3] 0.0-10.4 sec  2.88 MBytes  2.32 Mbits/sec
[ID] Interval       Transfer     Bandwidth
[ 3] 0.0-10.5 sec  3.25 MBytes  2.60 Mbits/sec


root@n4:/tmp# iperf -s -u -i 2
------------------------------------------------------------
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size:  160 KByte (default)
------------------------------------------------------------
[ 3] local 11.0.0.4 port 5001 connected with 20.0.0.100 port 54274
[ID] Interval     Transfer     Bandwidth      Jitter Lost/Total Datag
[ 3] 0.0- 2.0 sec 1.13 MBytes  4.76 Mbits/sec  1.149ms 75/884(8.5%)
[ 3] 0.0- 2.0 sec 3 datagrams received out-of-order
[ 3] 2.0- 4.0 sec 1.16 MBytes  4.85 Mbits/sec  0.842ms 68/893(7.6%)
[ 3] 2.0- 4.0 sec 2 datagrams received out-of-order
[ 3] 4.0- 6.0 sec 1.18 MBytes  4.96 Mbits/sec  1.403ms 47/891(5.3%)
[ 3] 4.0- 6.0 sec 3 datagrams received out-of-order
[ 3] 6.0- 8.0 sec 1.17 MBytes  4.91 Mbits/sec  0.780ms  56/891(6.3%)
[ 3] 8.0-10.0 sec 1.18 MBytes  4.95 Mbits/sec  1.160ms  52/893(5.8%)
[ 3] 0.0-10.0 sec 5.83 MBytes  4.89 Mbits/sec  1.360 ms297/4459(6.7%)
[ 3] 0.0-10.0 sec 9 datagrams received out-of-order
```

Se ve que hay un porcentaje de pérdida de un 7 % aprox. (6.7 %). El valor es bajo debido a que solo compite con TCP el cual se auto-regulará bajando la tasa de transmisión y dejando espacio para UDP.

8.1.1.3. Tests combinados a 10Mbps con QoS

Las siguientes pruebas realizadas muestran la configuración y resultados en el cual se marca y atiende el tráfico de acuerdo al campo Flow Label para brindar QoS. Este se considera el primer modelo de "QoS Extendida" presentado en el trabajo. En las pruebas realizadas en lugar de logear el tráfico como se mostró con los comandos de los ejemplos del capítulo anterior, en la sección bajo el título **"Herramienta de comparación"**, se marca a que flowID pertenecen los paquetes, para que luego sean manejado por el acondicionador de tráfico dentro de la red. En las muestras, debido a que el tráfico se genera desde el mismo origen, solo se compara el flowID. En un

esquema con emisores dispersos, donde se tiene diferentes hosts generando el tráfico, es importante comparar también las direcciones IPv6 de origen y destino.

Previo al manejo diferenciado se deben establecer los valores del/de los campo/s y el tratamiento. Esto se puede realizar de forma compleja y escalable como podría ser mediante un protocolo de señalización o de forma simple con una configuración manual. Por ejemplo, se puede tener pre-acordado los valores de Flow Label y su tratamiento. En el caso de los ejemplos, como se indica en la introducción del capítulo, se usa un mecanismo manual donde se mantienen y configuran las siguientes reglas:

- Solo se aplica tratamiento diferenciado sobre los enlaces saturados. En este caso solo en el enlace de 10Mbps.
- Los flujos sin marcar o con el valor default, se encolan en una clase con un límite de 512Kbps.
- Los flujos con valores de Flow Label menores a 1024, distintos del default cero (0), deben recibir un tratamiento separado con un límite de ancho de banda de 1Mbps. En este ejemplo será el tráfico UDP.
- Los flujos TCP, solicitan cada uno un cola separada, indicando explícitamente el ancho de banda digital requerido, en estos ejemplos uno con 3Mbps y otro con 5Mbps. Se usa el campo Flow Label para indicar el piso solicitado.

Estas reglas para el ejemplo específico con los parámetros extras de números de puertos destinos y protocolos es la mostrada a continuación en la lista de flujos:

- Flow 1: UDP:8000 con un valor de FL (Flow Label) 10 ó 0x0000A va a una cola separada de 1Mbps como máximo y se intenta asegurar 1Mbps. El tráfico generado tiene un máximo de 5Mbps.
- Flow 2: TCP:8001 codificará el valor FL 5000 ó 0cx01388, va a cola de 5Mbps con borrow, es decir puede sobre-pasarse, si el resto del tráfico no usa todo el ancho de banda lo podrá utilizar.
- Flow 3: TCP:8002 codificará el valor FL 2000 ó 0x0007D0, va a cola de 2MBps, sin borrow, tiene como techo y piso los 2Mbps.
- Flow 4: TCP:5001 sin marcar va a cola default de 512Kbps.

Valor FL usado	Rango	Rango de BW	reglas
0x0000a	1..1023	1..1Mbps	1:10 htb rate 1mbit ceil 1mbit burst 15k
0x00000	0000	512Kbps	1:40 htb rate 512kbit (default)
0x01388	5000	5..10Mbps	1:20 htb rate 5mbit ceil 10mbit burst 15k
0x007d0	2000	2..2Mbps	1:30 htb rate 2mbit ceil 2mbit burst 15k

Cuadro 8.1: Valores Flow Label y tratamiento asociado para el ejemplo con QoS

Los comandos a ejecutar en los equipos intermedios para configurar las reglas son los que se presentan a continuación. Primero se configuran las reglas de clasificación de flujos de acuerdo a la marca de Flow Label. Estas marcas son internas al kernel de GNU/Linux y no van "escritas" en los paquetes.

```
root@n1#
ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x000FFFFF=0x0000A"
 -j MARK --set-mark 1
ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x000FFFFF=0x01388"
 -j MARK --set-mark 2
ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x000FFFFF=0x007D0"
 -j MARK --set-mark 3

root@n1# ip6tables -t mangle -L -n -v
...
Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source     destination
    0     0 MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0xfffff=0x7d0" MARK set 0x3
    0     0 MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0xfffff=0x1388" MARK set 0x2
    0     0 MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0xfffff=0xa" MARK set 0x1
```

La regla que compara contra el valor 0xA podría escribirse de forma que respete la política indicando todo el tráfico con valor menor a 1024.

```
root@n1#
ip6tables -I POSTROUTING -t mangle -m u32
 --u32 "0&0x000FFFFF=0x1:0x003F -j MARK --set-mark 1
```

Luego se configura el tratamiento y encolado de acuerdo a las marcas internas generadas por los comandos anteriores. En este caso se usan colas de tipo HTB, con *borrow* para la identificada con el 1:20, pudiendo llegar hasta 10Mbps, luego el resto de las colas tienen un piso o mínimo y un máximo o techo, ver tabla 8.1 (1:20 tiene un techo de 10Mbps que es el máximo del enlace, por lo tanto no se lo considera como tal).

```
root@n1#
tc qdisc add dev eth1 root handle 1: htb default 40
tc class add dev eth1 parent 1: classid 1:1 htb rate 10mbit
tc class add dev eth1 parent 1:1 classid 1:10 htb rate 1mbit
         ceil 1mbit burst 15k
```

```
tc class add dev eth1 parent 1:1 classid 1:20 htb rate 5mbit
         ceil 10mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:30 htb rate 2mbit
         ceil  2mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:40 htb rate 512kbit
```

Para ver como quedan las colas o clases creadas se ejecuta el comando:

```
root@n1# tc -s class show dev eth1
class htb 1:1 root rate 10000Kbit ceil 10000Kbit
 burst 1600b cburst 1600b
 Sent 704 bytes 8 pkt (dropped 0, overlimits 0 requeues 0)
 rate 160bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 18813 ctokens: 18813

class htb 1:10 parent 1:1 prio 0 rate 1000Kbit ceil 1000Kbit
 burst 15Kb cburst 1600b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 1920000 ctokens: 200000

class htb 1:20 parent 1:1 prio 0 rate 5000Kbit ceil 10000Kbit
 burst 15Kb cburst 1600b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 384000 ctokens: 20000

class htb 1:30 parent 1:1 prio 0 rate 2000Kbit ceil 2000Kbit
 burst 15Kb cburst 1600b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 960000 ctokens: 100000

class htb 1:40 parent 1:1 prio 0 rate 512000bit ceil 512000bit
 burst 1600b cburst 1600b
 Sent 704 bytes 8 pkt (dropped 0, overlimits 0 requeues 0)
 rate 160bit 0pps backlog 0b 0p requeues 0
 lended: 8 borrowed: 0 giants: 0
 tokens: 367188 ctokens: 367188
```

A continuación se arma el mapeador desde las marcas generadas por el clasificador con los valores de FlowID a las diferentes colas. Se podría considerar como el planificador, scheduler o filtro de la arquitectura. Se configura en el mismo equipo de las clases o colas.

```
root@n1#
tc filter add dev eth1 parent 1: protocol ipv6 handle 1 fw flowid 1:10
tc filter add dev eth1 parent 1: protocol ipv6 handle 2 fw flowid 1:20
tc filter add dev eth1 parent 1: protocol ipv6 handle 3 fw flowid 1:30

root@n1# tc -s filter show dev eth1
filter parent 1: protocol ipv6 pref 49150 fw
filter parent 1: protocol ipv6 pref 49150 fw handle 0x3 classid 1:30
filter parent 1: protocol ipv6 pref 49151 fw
filter parent 1: protocol ipv6 pref 49151 fw handle 0x2 classid 1:20
filter parent 1: protocol ipv6 pref 49152 fw
filter parent 1: protocol ipv6 pref 49152 fw handle 0x1 classid 1:10
```

Luego de la configuración se pueden ejecutar algunos tests de forma individual para ver que las colas seleccionadas son las adecuadas. Por ejemplo, el primero muestra el encolado en la default, de 512Kbps.

```
root@n5:/tmp# iperf -V -c 2001:db8:4::1
------------------------------------------------------------
Client connecting to 2001:db8:4::1, TCP port 5001
TCP window size: 20.8 KByte (default)
------------------------------------------------------------
[  3] local 2001:db8:20::100 port 53730 connected with 2001:db8:4::1
      port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-12.4 sec  1.00 MBytes  679 Kbits/sec

root@n4:/tmp# iperf -s -V
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 2001:db8:4::1 port 5001 connected with 2001:db8:20::100
      port 53730
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-17.4 sec  1.00 MBytes  482 Kbits/sec

root@n1# tc -s class show dev eth1 | grep Sent
 Sent 3343744 bytes 2300 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 3343744 bytes 2300 pkt (dropped 0, overlimits 0 requeues 0)
```

El tráfico marcado con valor menor que 1024. En este caso se usa TCP, luego UDP. Va a la cola 1:10 de 1Mbps.

```
root@n5:/tmp# ./tcp6-socky -C -r -X 10 -d 2001:db8:4::1 -E K 0 1400 10
RBUF:      87380
SBUF:      16384
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Trying to connect to 2001:db8:4::1:8000
Connected to 2001:db8:4::1:8000
RBUF:      87380
SBUF:      21280
MSS:       1428
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Extra Options: opt=K times=0 delay=0 chunk=1400 rchar=A tmout=10

proc 1582000 bytes in 10000487.000000 usec or 10.000487 sec
BW=1.265538 Mbps
 Return: 0

root@n4:/tmp# ./tcp6-socky -L -r -E k
RBUF:      87380
SBUF:      16384
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Waiting connection on 0:::8000 backlog: 1 model: 0
Connected from 2001:db8:20::100:59508
Extra Options: opt=k times=0 delay=0 chunk=8192 rchar=A tmout=0
```

```
proc 1582000 bytes in 13452145.000000 usec or 13.452145 sec
BW=0.940817 Mbps
 Return: 0
Waiting connection on 0:::8000 backlog: 1 model: 0

root@n1# tc -s class show dev eth1 | grep Sent
 Sent 6030400 bytes 4136 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 1569614 bytes 1041 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 4460786 bytes 3095 pkt (dropped 0, overlimits 0 requeues 0)
```

Probando con UDP y generando tráfico de 2Mbps aprox.

```
root@n5:/tmp# time ./udp6-socky -S 2001:db8:4::1 -d 5000 -X 10 -l 1400
              -c 2000
RBUF:       163840
SBUF:       163840
MCAST:      0
REUSE:      0
MTUDISC:    1

real 0m11.072s
user 0m0.000s
sys 0m0.364s
...
```

Luego de varios tests y monitoreando con `ethstatus(1)`. En este caso se ve que se genera 122KBps de tráfico, aprox. 1Mbps: $122 \times 8bits = 976Mbps$.

```
root@n1# tc -s class show dev eth1 | grep Sent
 Sent 37597042 bytes 25871 pkt (dropped 0, overlimits 0 requeues 0)
Sent 33122498 bytes 22623 pkt (dropped 58261, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 4474544 bytes 3248 pkt (dropped 0, overlimits 0 requeues 0)

root@n1# ethstatus -i eth1 -v mono

...
         ###
        ON/OFF                        RX  TX
                           122.78 KB/s   88 Packets/s
...
```

A continuación los tests individuales para las clases TCP de 5 y 2Mbps, la primera con borrow y la segunda con limitación de máximo. Flujo con borrow y sin competencia usa el máximo (se encola en 1:20).

```
root@n5:/tmp# ./tcp6-socky -C -r -X 5000 -d 2001:db8:4::1 -p 8001
              -E K 0 1400 10
...
proc 11029200 bytes in 10000881.000000 usec or 10.000881 sec
BW=8.822583 Mbps
 Return: 0

root@n1# tc -s class show dev eth1 | grep Sent
 Sent 58167664 bytes 39754 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 41697128 bytes 28488 pkt (dropped 58396, overlimits 0 requeues 0)
 Sent 11988562 bytes 7933 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 4481974 bytes 3333 pkt (dropped 0, overlimits 0 requeues 0)
```

Flujo TCP sin borrow, con máximo o techo.

```
root@n5:/tmp# ./tcp6-socky -C -r -X 2000 -d 2001:db8:4::1 -p 8002
              -E K 0 1400 10
...
proc 2865800 bytes in 10000380.000000 usec or 10.000380 sec
BW=2.292553 Mbps
 Return: 0

root@n1# tc -s class show dev eth1 | grep Sent
 Sent 61097420 bytes 41714 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 41697128 bytes 28488 pkt (dropped 58396, overlimits 0 requeues 0)
 Sent 11988562 bytes 7933 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 2928098 bytes 1941 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 4483632 bytes 3352 pkt (dropped 0, overlimits 0 requeues 0)
```

Los matching se pueden ver también en el clasificador.

```
root@n1# ip6tables -t mangle -L -n -v
...

Chain POSTROUTING (policy ACCEPT 107K packets, 146M bytes)
 pkts bytes target     prot opt in     out     source     destination
 1941 2901K MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0xfffff=0x7d0" MARK set 0x3
 7933   12M MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0xfffff=0x1388" MARK set 0x2
86884  126M MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0xfffff=0xa" MARK set 0x1
```

Una vez realizados los tests individuales y teniendo la certeza que el tráfico se encola en la clase correspondiente se realizan las pruebas combinadas (con competencia), previamente de haber puesto a cero los contadores. La generación de tráfico con los cuatro flujos es la siguiente.

```
# iperf -V -c 2001:db8:4::1 &
./tcp6-socky -C -r -X 2000 -d 2001:db8:4::100 -p 8002 -E K 0 1400 13 &
./tcp6-socky -C -r -X 5000 -d 2001:db8:4::100 -p 8001 -E K 0 1400 13 &
time ./udp6-socky -S 2001:db8:4::50 -d 1500 -X 10 -l 1400 -c 4200
```

Los resultados son los esperados, el más prioritario dio un poco más de 5Mbps, luego los restantes de acuerdo a lo configurado, 2Mbps, 1Mbps y 512Kbps.

```
root@n4:/tmp# ./tcp6-socky -L -p 8002 -r -E k
...
proc 3496388 bytes in 14886256.000000 usec or 14.886256 sec
BW=1.878989 Mbps
...

root@n4:/tmp# ./tcp6-socky -L -p 8001 -r -E k
...
proc 10135188 bytes in 13669201.000000 usec or 13.669201 sec
BW=5.931693 Mbps
...

root@n4:/tmp# iperf -s -V
...
[  4]  0.0-17.4 sec  1.00 MBytes   482 Kbits/sec

root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w fl-n1-v2.pcap
```

Analizando la gráfica a partir del tráfico capturado en la figura 8.7 se ve que cada tráfico obtiene lo acordado. El tráfico TCP más importante en verde, el siguiente en rojo, el UDP en negro y el sin marcar (default) en azul, respetando los colores de los ejemplos anteriores.

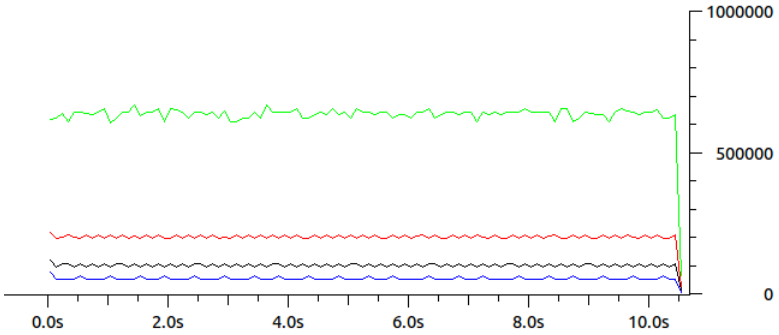


Figura 8.7: Test combinado aplicando QoS

8.1.1.4. Tests a 10Mbps con QoS, otro ejemplo

En este otro ejemplo los flujos TCP que marcan el Flow Label solicitan cada uno un cola separada con los mismos parámetros de QoS para este caso con 4Mbps de mínimo y 8Mbps de máximo, el resto del tráfico recibe el servicio en un clase común de 1Mbps que auspicia de default. En el ejemplo se diferencian los tráficos, incluso pidiendo la misma QoS para que no compitan entre sí.

- Flow 1: UDP:8000 con un valor de FL(Flow Label) de 10 ó 0x0000A va a una cola separada de 1Mbps.
- Flow 2: TCP:8001 con el valor 4000 ó 0x00FA0, va a una cola de 4..8Mbps.
- Flow 3: TCP:8002 con el valor 4001 ó 0x00FA1, va a otra de cola de 4..8MBps.

La configuración para lograr este comportamiento se presenta a continuación.

```
root@n1#
ip6tables -t mangle -F
ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x000FFFFF=0x0000A"
 -j MARK --set-mark 1
ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x000FFFFF=0x00FA0"
 -j MARK --set-mark 2
ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x000FFFFF=0x00FA1"
 -j MARK --set-mark 3

root@n1#
tc qdisc del dev eth1 root handle 1: htb
tc qdisc add dev eth1 root handle 1: htb default 40
tc class add dev eth1 parent 1: classid 1:1 htb rate 10mbit
tc class add dev eth1 parent 1:1 classid 1:10 htb rate 1mbit
   ceil 1mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:20 htb rate 4mbit
   ceil 8mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:30 htb rate 4mbit
   ceil 8mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:40 htb rate 512kbit

root@n1#
tc filter add dev eth1 parent 1: protocol ipv6 handle 1 fw flowid 1:10
```

```
tc filter add dev eth1 parent 1: protocol ipv6 handle 2 fw flowid 1:20
tc filter add dev eth1 parent 1: protocol ipv6 handle 3 fw flowid 1:30
```

Los filtros instalados se muestran en el texto a continuación.

```
root@n1# tc -s filter show dev eth1
filter parent 1: protocol ipv6 pref 49150 fw
filter parent 1: protocol ipv6 pref 49150 fw handle 0x3 classid 1:30
filter parent 1: protocol ipv6 pref 49151 fw
filter parent 1: protocol ipv6 pref 49151 fw handle 0x2 classid 1:20
filter parent 1: protocol ipv6 pref 49152 fw
filter parent 1: protocol ipv6 pref 49152 fw handle 0x1 classid 1:10
```

Por último se ejecutan las pruebas del lado de los clientes.

```
root@n5:/tmp#
iperf -V -c 2001:db8:4::1 &

...# ./tcp6-socky -C -r -X 4000 -d 2001:db8:4::100 -p 8002 -E K 0 1400 13

...# ./tcp6-socky -C -r -X 4001 -d 2001:db8:4::100 -p 8001 -E K 0 1400 13

...# time ./udp6-socky -S 2001:db8:4::50 -d 1500 -X 10 -l 1400 -c 4200
```

Y del lado del servidor.

```
root@n4:/tmp#
iperf -s -V

...# ./tcp6-socky -L -p 8001 -r -E k

...# ./tcp6-socky -L -p 8002 -r -E k
```

Capturando el tráfico para luego analizarlo.

```
root@n1/tmp# tcpdump -n -i eth1 -s 100 -w fl-n1-4000-v2.pcap
```

Las salidas de los clientes se muestran en los textos siguientes.

```
root@n4:/tmp# ./tcp6-socky -L -p 8002 -r -E k
RBUF:      87380
SBUF:      16384
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Waiting connection on 0:::8002 backlog: 1 model: 0
Connected from 2001:db8:20::100:41612
Extra Options: opt=k times=0 delay=0 chunk=8192 rchar=A tmout=0

proc 6941928 bytes in 13998236.000000 usec or 13.998236 sec
BW=3.967316 Mbps
 Return: 0
Waiting connection on 0:::8002 backlog: 1 model: 0
```

Para el otro cliente.

```
root@n4:/tmp# ./tcp6-socky -L -p 8001 -r -E k
RBUF:      87380
SBUF:      16384
MSS:       536
REUSE:     1
NODELAY:   0
KEEPALIVE: 0
Waiting connection on 0:::8001 backlog: 1 model: 0
Connected from 2001:db8:20::100:50465
Extra Options: opt=k times=0 delay=0 chunk=8192 rchar=A tmout=0

proc 6759872 bytes in 13822108.000000 usec or 13.822108 sec
BW=3.912498 Mbps
 Return: 0
Waiting connection on 0:::8001 backlog: 1 model: 0
```

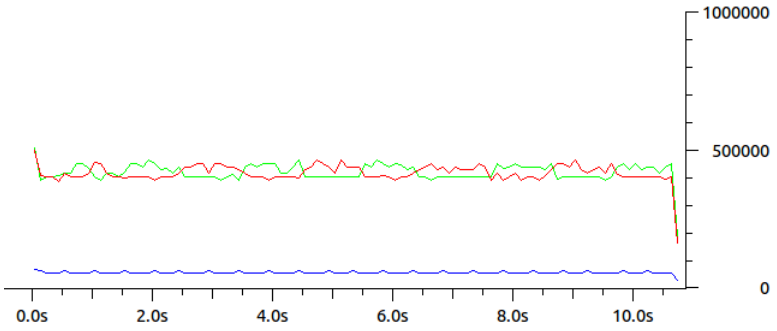


Figura 8.8: Otro ejemplo, tests combinados aplicando QoS

Para el tráfico default.

```
root@n4:/tmp# iperf -s -V
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 2001:db8:4::1 port 5001 connected with 2001:db8:20::100
      port 42865
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-15.2 sec   896 KBytes   482 Kbits/sec
```

La gráfica 8.8 muestra los resultados de los anchos de banda obtenidos.

8.1.2. Conclusiones del primer modelo

El modelo muestra que mediante una asignación específica del campo se puede ofrecer un manejo diferenciado para los diferentes flujos. En los primeros tests se muestra que si no tienen competencia los flujos no se generan problemas, pero cuando se combinan pugnando por recursos similares, en este caso el ancho de banda (que es de "escasos" 10Mbps) los resultados obtenidos no son buenos. Al agregarle el marcado para obtener QoS se logra el resultado diferenciado, teniendo los flujos la QoS solicitada. En este caso sólo se usa el valor del campo Flow Label para diferenciar el manejo separado. En un escenario donde la cantidad de flujos y datos en curso es muy grande, como sucede con Internet, como ya se mencionó, se debe agregar el "matching" de las direcciones origen y destino.

De acuerdo a lo estudiado en el capítulo anterior, la siguiente propuesta sigue algunos lineamientos del documento que especifica el uso del campo: RFC-6437 [AJCR11]. Por ejemplo:

- Se utiliza un valor distinto de cero.

- El valor debe ser marcado por el origen y pretende no ser modificado en curso.

Pero tiene algunas cuestiones contrarias a lo estipulado por IETF, como:

- El modelo propuesto requiere el mantenimiento de información a modo de estados o conocimiento previo, necesario para el tratamiento de los flujos de acuerdo a la marca del campo.
- En el modelo propuesto los valores de los campos de la 3-tupla: dir. origen, dir. destino, Flow Label; podrían usarse para identificar que paquetes pertenecen a un flujo, aunque no es condición necesaria y podría no respetarse. En la asignación no se fuerza que los valores no se puedan repetir. Esta cuestión puede lograrse al usar la API ofrecida por GNU/Linux. En los ejemplos el control depende de los usuarios.
- Los routers/nodos intermedios están dependiendo del valor del Flow Label para determinar el tratamiento.
- No se utilizan valores (pseudo)-aleatorios, en su lugar se usan valores específicos.

Una cuestión que se relaja en la nueva RFC y que no es mirada en la propuesta:

- Los nodos IPv6 no deben asumir ninguna propiedad matemática u otra sobre el valor del campo.

En la implementación propuesta el valor específico del campo indica el tratamiento diferenciado de acuerdo a la QoS. Como no se exige no se considera la característica de asumir propiedades diferentes acorde a su valor.

8.2. Propuesta del uso del campo Flow Label para algoritmos de Control de Congestión

La segunda propuesta es ofrecer un modelo donde los flujos TCP que compiten por los recursos de red, en un ambiente donde los algoritmos de control de congestión (CC) varían, puedan recibir un tratamiento equitativo. Una alternativa es tratar en clases separadas cada flujo, distinguibles a partir de la 3-tupla: FlowID, Dir. Origen, Dir. Destino. En este caso cada flujo TCP tendría su ancho de banda sin competir directamente con los demás. Para este tipo de manejo el CC no dependería de la competencia, dada la situación que cada uno maneje su algoritmo de CC, podría

disponer del ancho de banda dado por el manejador de colas dentro del dominio de QoS. En principio es un sistema FQ, como ya se vio en el capítulo **Estrategias de Encolado**, en cual la distinción de flujos se realiza por la 3-tupla que considera solo campos del encabezado IPv6, incluyendo el campo Flow Label o Flow ID. Por supuesto que este necesita colaboración de los extremos generadores de tráfico para marcar y distinguir los flujos. En el caso que no se realizara esta distinción el comportamiento va a ser como el mostrado en el análisis realizado en torno a la figura 8.5, donde todos compiten por todo los recursos y si existen algoritmos que tienen diferentes CC, como el tráfico UDP que no se auto-regula, podría existir una problema de desigualdad al compartir el ancho de banda.

8.2.1. Tests realizados, segunda propuesta

Con el objetivo de demostrar la utilidad del campo Flow Label para el segundo modelo intentando brindar QoS se utiliza la misma topología de los test del primer modelo, ya presentada en la figura 8.1.

8.2.1.1. Tests combinados a 10Mbps sin QoS para diferenciar CC

Los tests combinados sin configurar QoS obtendrán resultados como los de la figura 8.5 en la sección **Tests combinados a 10Mbps sin QoS**. A estas pruebas se puede agregar otro ejemplo donde el algoritmo de control de congestión usado es pro-activo como el caso de TCP Vegas [BP95], a diferencia de los re-activos como son TCP Reno o TCP Cubic.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w test-comb-m2-sin-qos.pcap

root@n5:/tmp# ./tcp6-socky -C -r -X 2000 -d 2001:db8:4::100 -p 8002
                           -G vegas -E K 0 1400 13 &
              ./tcp6-socky -C -r -X 5000 -d 2001:db8:4::100 -p 8001
                           -E K 0 1400 13 &
              time ./udp6-socky -S 2001:db8:4::50 -d 1500 -X 10 -l 1400
                                -c 4200

root@n4:/tmp# ./tcp6-socky -L -p 8001 -r -E k &
              ./tcp6-socky -L -p 8002 -r -E k
```

Los resultados se pueden observar en la figura 8.9. En la misma se ve que, además de los picos dados por la competencia de los tráficos, para el caso de agregar un flujo que utiliza el algoritmo de control de congestión Vegas (dibujado de color verde), este se relega con respecto al flujo TCP con Cubic o Reno.

TCP Vegas [BP95] tiene ventajas conocidas por la forma que realiza el control de congestión, pero como se ve en los resultados también en determinados escenarios se

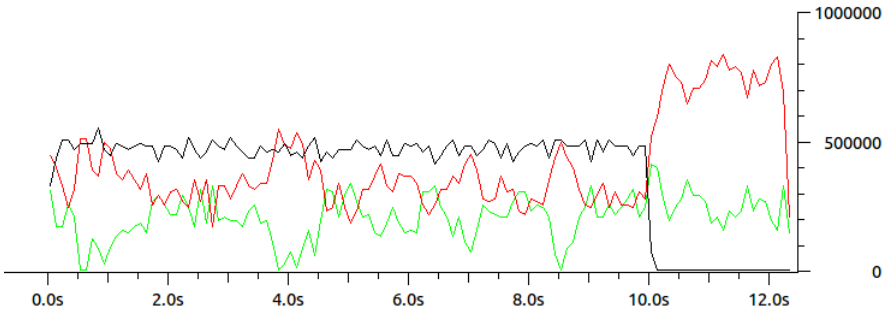


Figura 8.9: Tests combinados, diferentes CC y UDP sin QoS

encuentran resultados adversos. Esta situación es estudiada y presentada tempranamente en el documento citado: [MLAW99], donde en las conclusiones se indica que Vegas alcanza mejor rendimiento que Reno aunque cuando compite con otras conexiones Reno, se ve penalizado por la "naturaleza agresiva de TCP Reno". Se indica en el artículo la necesidad de estudiar formas de aislar Vegas de Reno ya que "no cooperan" entre sí. Reno, o mejor mencionado como New Reno fue definido por Van Jacobson en [Jac] y luego estandarizado por los documentos IETF como RFC-2001 y correcciones como RFC-5681.

Las desventajas de Vegas se analizan también en artículos como [HBG00] donde se menciona que los mecanismos de control de congestión exhiben problemas al competir con otras conexiones.

En [FS03], si bien se menciona la ventaja de Vegas ejecutando en un entorno aislado sobre TCP Reno, este, cuando tiene que compartir enlaces con recursos limitados es sobrepasado por Reno, el cual "roba" parte del ancho de banda dando mejores resultados.

En otros artículos como [FV03] se estudian adaptaciones de los parámetros, α y β, para que esta situación, de pérdida de rendimiento, no suceda con el algoritmo Vegas y se menciona la mala productividad que este obtiene si compite directamente contra Reno o algoritmos de control congestión similares que trabajan de forma re-activa.

Luego de varios estudios realizados por diferentes investigaciones se han encontrado problemas en Vegas y se han propuestos diferentes modificaciones para solucionarlos. Por ejemplo, en el artículo [SJA05] se mencionan algunos de los problemas descubiertos. Además de la competencia con Reno, se remarca el bajo rendimiento ante el re-ruteo de las sesiones, con enlaces asimétricos o ante espacios pequeños en los buffers de los routers intermedios. El mismo documento desarrolla una modificación sobre el algoritmo llamado "Vegas-A", para dar soluciones a los inconvenientes. Como

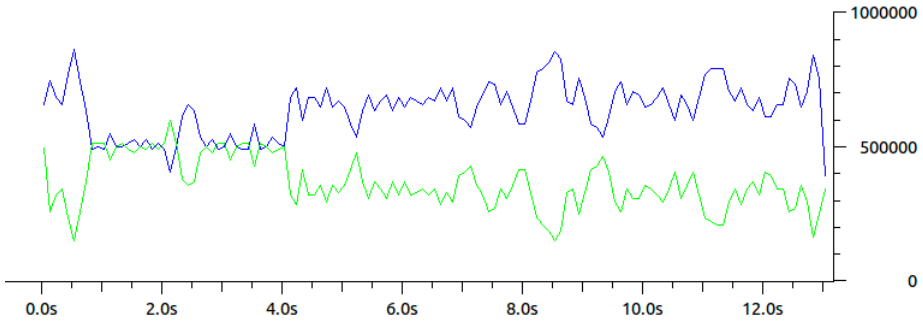


Figura 8.10: Tests combinados, Cubic (azul) vs Vegas (verde) sin QoS

esta, existen otras propuestas, por ejemplo, “Stabilized Vegas” [CL03], “TCP New Vegas” [SS05] o “TCP Vegas-V” [ZXWZ12].

En este trabajo se aborda una posible solución sin modificar el módulo de TCP-Vegas, sino tratando de forma diferente el tráfico con el algoritmo de CC. Si se realizan los tests comparando la competencia entre Reno y Vegas, o Cubic con Vegas, pero sin incluir competencia de tráfico UDP, también se puede re-corroborar el comportamiento mencionado.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w
              test-comb-m2-sin-qos-sin-udp-cubic-vegas.pcap

root@n5:/tmp# ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8002
                           -G vegas -E K 0 1400 13 &
              ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8001
                           -G cubic -E K 0 1400 13
```

En el gráfico 8.10 se ve como Vegas pierde algo de terreno frente a Cubic.
Similar sucede si se lo compara con Reno en competencia de acuerdo al gráfico 8.11.

```
root@n1:/tmp# tcpdump -n -i eth1 -s 100 -w
              test-comb-m2-sin-qos-sin-udp-reno-vegas.pcap

root@n5:/tmp# ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8002
                           -G vegas -E K 0 1400 13 &
              ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8001
                           -G reno -E K 0 1400 13
```

8.2.1.2. Tests combinados a 10Mbps con QoS

Se configuran los tests para que el sistema de encolado trate de darle la mitad del ancho de banda disponible a las sesiones que usan Vegas como algoritmo de CC vs. otros re-activos, como Reno o Cubic. La distinción se realiza mediante el marcado de tráfico en el Flow Label de IPv6. La configuración de prueba en el ejemplo es: Vegas lleva los valores de Flow Label con los dos bits menos significantes, LSb, con un valor de 1 (uno), es decir 01_2. Los flujos que usan algoritmos re-activos para el CC llevarán en estos mismos bits el valor 2 (dos), es decir 10_2. Para el trafico sin marcar estarán en 0. A continuación se muestra una posible configuración:

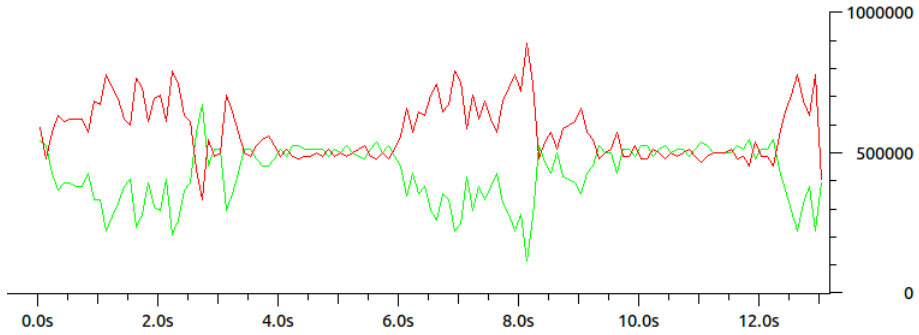


Figura 8.11: Tests combinados, Reno (rojo) vs Vegas (verde) sin QoS

```
root@n1:/tmp#

ip6tables -t mangle -F
ip6tables -t filter -F
tc qdisc add dev eth1 root tbf rate 10mbit latency 0.5ms burst 20KB

ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x00000003=0x00001"
          -j MARK --set-mark 1
ip6tables -I POSTROUTING -t mangle -m u32 --u32 "0&0x00000003=0x00002"
          -j MARK --set-mark 2

ip6tables -t mangle -L -n -v

root@n1:/tmp# ip6tables -t mangle -L -n -v | grep -A10 POSTROUT
Chain POSTROUTING (policy ACCEPT 28 packets, 2932 bytes)
 pkts bytes target     prot opt in     out     source     destination
    0     0 MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0x3=0x2" MARK set 0x2
    0     0 MARK       all      *      *       ::/0       ::/0
    u32 "0x0&0x3=0x1" MARK set 0x1

tc qdisc del dev eth1 root tbf rate 10mbit latency 0.5ms burst 20KB

tc qdisc add dev eth1 root handle 1: htb default 40
tc class add dev eth1 parent 1: classid 1:1 htb rate 10mbit
tc class add dev eth1 parent 1:1 classid 1:10 htb rate 5mbit
         ceil 10mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:20 htb rate 5mbit
         ceil 10mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:40 htb rate 512kbit
```

Para ver el estado de la clases configuradas se ejecuta:

```
root@n1:/tmp# tc -s class show dev eth1
class htb 1:1 root rate 10000Kbit ceil 10000Kbit burst 1600b cburst 1600b
 Sent 176 bytes 2 pkt (dropped 0, overlimits 0 requeues 0)
 rate 64bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 18813 ctokens: 18813

class htb 1:10 parent 1:1 prio 0 rate 5000Kbit ceil 10000Kbit burst
      15Kb cburst 1600b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 384000 ctokens: 20000

class htb 1:20 parent 1:1 prio 0 rate 5000Kbit ceil 10000Kbit burst
 15Kb cburst 1600b
```

```
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 384000 ctokens: 20000

class htb 1:40 parent 1:1 prio 0 rate 512000bit ceil 512000bit burst
 1600b cburst 1600b
 Sent 176 bytes 2 pkt (dropped 0, overlimits 0 requeues 0)
 rate 64bit 0pps backlog 0b 0p requeues 0
 lended: 2 borrowed: 0 giants: 0
 tokens: 367188 ctokens: 367188
```

Luego se agregan los filtros:

```
root@n1:/tmp#
tc filter add dev eth1 parent 1: protocol ipv6 handle 1 fw flowid 1:10
tc filter add dev eth1 parent 1: protocol ipv6 handle 2 fw flowid 1:20


root@n1:/tmp# tc -s filter show dev eth1
filter parent 1: protocol ipv6 pref 49151 fw
filter parent 1: protocol ipv6 pref 49151 fw handle 0x2 classid 1:20
filter parent 1: protocol ipv6 pref 49152 fw
filter parent 1: protocol ipv6 pref 49152 fw handle 0x1 classid 1:10

root@n1:/tmp# tc -s class show dev eth1 | grep Sent
 Sent 1760 bytes 20 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 1760 bytes 20 pkt (dropped 0, overlimits 0 requeues 0)
```

En este paso se ejecutan los tests del lado del servidor y del cliente, pero de forma individual para corroborar que el encolado es el apropiado. Se ve que alcanza casi el máximo de la capacidad disponible.

```
root@n4:/tmp#
./tcp6-socky -L -p 8001 -r -E k &
./tcp6-socky -L -p 8002 -r -E k &

root@n5:/tmp# ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8002 -X 8002
              -G vegas -E K 0 1400 13

...
proc 13752200 bytes in 13000386.000000 usec or 13.000386 sec
BW=8.462641 Mbps
 Return: 0

root@n1:/tmp# tc -s class show dev eth1 | grep Sent
 Sent 14584264 bytes 9666 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 14581374 bytes 9633 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 2890 bytes 33 pkt (dropped 0, overlimits 0 requeues 0)
```

Luego probando con Cubic sucede lo mismo.

```
root@n5:/tmp# ./tcp6-socky -C -r -d 2001:db8:4::50 -p 8001 -X 8001
             -G cubic -E K 0 1400 13
...
proc 14151200 bytes in 13001016.000000 usec or 13.001016 sec
BW=8.707750 Mbps
 Return: 0

root@n1:/tmp# tc -s class show dev eth1 | grep Sent
 Sent 29591214 bytes 19605 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 15005996 bytes 9928 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 14581374 bytes 9633 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 3844 bytes 44 pkt (dropped 0, overlimits 0 requeues 0)
```

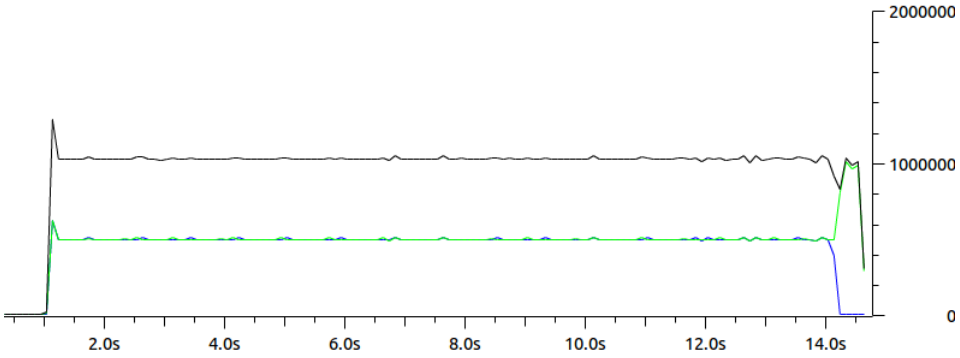


Figura 8.12: Tests combinados, Cubic (azul) vs Vegas (verde) y total (negro) con QoS

Se puede observar en la salida de los comandos, que, de acuerdo al valor del Flow Label se encolaron en las colas para el propósito y que al no existir competencia utilizan el borrow y aprovechan la capacidad máxima del enlace.

Por último, luego de los tests individuales, se realizan los combinados compitiendo Vegas con Cubic o Reno, pero con la QoS configurada. El gráfico de los resultados obtenidos se muestra en la figura 8.12 donde se puede ver que el ancho de banda es dividido de forma equitativa.

```
root@n5:/tmp# ./tcp6-socky -C -r -d 2001:db8:4::50 -p 8001 -X 8001 -G cubic
              -E K 0 1400 23 &
              ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8002 -X 8002 -G vegas
              -E K 0 1400 23
...
proc 13559000 bytes in 23000756.000000 usec or 23.000756 sec
...
proc 14033600 bytes in 23000907.000000 usec or 23.000907 sec
BW=4.716019 Mbps
BW=4.881060 Mbps
...
```

Si se realizan los tests agregando más de un flujo, uno más para cada tipo de algoritmo de CC el uso de ancho de banda total por grupo se mantendrá en la mitad de acuerdo a la figura: 8.13, pero luego dentro de cada grupo el desempeño muestra que Vegas puede manejarse de forma activa, figura 8.14, mientras que Cubic lo hace con los picos altos y bajos, figura 8.15. Se puede ver también la ventaja que toma la primera conexión de Vegas sobre la segunda.

```
./tcp6-socky -L -p 8001 -r -E k &
./tcp6-socky -L -p 8002 -r -E k &
./tcp6-socky -L -p 8003 -r -E k &
./tcp6-socky -L -p 8004 -r -E k &

root@n5:/tmp# ./tcp6-socky -C -r -d 2001:db8:4::50 -p 8001 -X 8001
              -G cubic -E K 0 1400 13 &
```

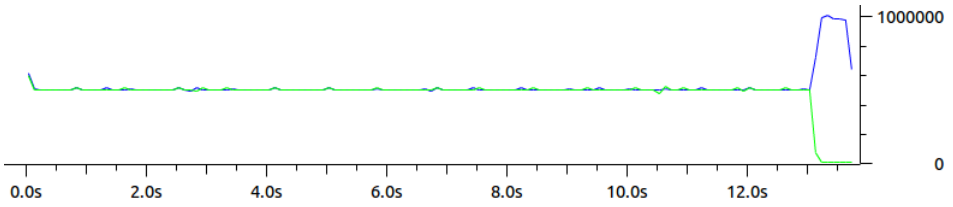


Figura 8.13: Tests combinados, 2 sesiones Cubic (azul) vs 2 Vegas (verde) con QoS

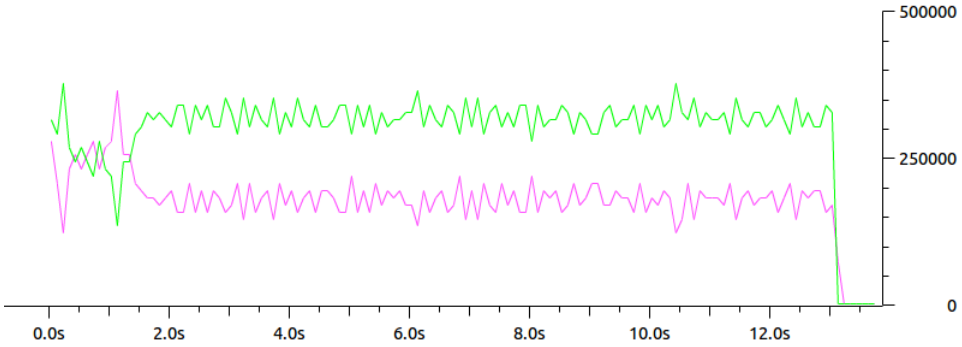


Figura 8.14: Tests combinados, 2 sesiones Cubic vs 2 Vegas, solo se grafica el comportamiento de Vegas

```
./tcp6-socky -C -r -d 2001:db8:4::100 -p 8002 -X 8002
-G vegas -E K 0 1400 13 &
./tcp6-socky -C -r -d 2001:db8:4::50 -p 8003 -X 1113
-G cubic -E K 0 1400 13 &
./tcp6-socky -C -r -d 2001:db8:4::100 -p 8004 -X 3402
-G vegas -E K 0 1400 13

...
Vegas
...
proc 4012400 bytes in 13000520.000000 usec or 13.000520 sec
BW=2.469070 Mbps
 Return: 0
...
Cubic
...
proc 4827200 bytes in 13000793.000000 usec or 13.000793 sec
BW=2.970403 Mbps
...
Vegas
...
proc 2881200 bytes in 13002333.000000 usec or 13.002333 sec
BW=1.772728 Mbps
...
Cubic
...
proc 4426800 bytes in 13000425.000000 usec or 13.000425 sec
BW=2.724096 Mbps
```

Si se agregan más sesiones en este escenario: tres para Cubic y tres para Vegas el enlace se comparte de forma pareja entre los dos grupos de algoritmos. Cubic en

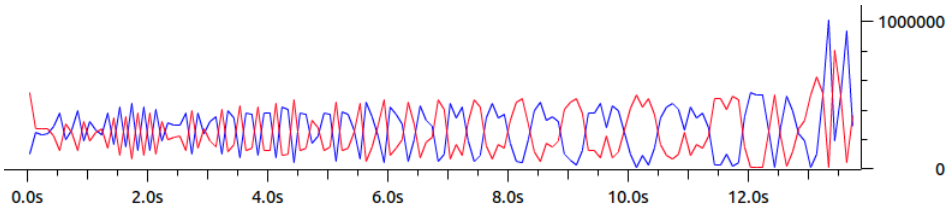


Figura 8.15: Tests combinados, 2 sesiones Cubic vs 2 Vegas, solo se grafica el comportamiento de Cubic

el control de congestión se comporta como es de esperar, pero Vegas ya cuando la competencia dentro de su grupo crece comienza a tener picos altos y caídas tendiendo a funcionar como un algoritmo re-activo y no pro-activo como debiera.

```
root@n4:/tmp#
 ./tcp6-socky -L -p 8001 -r -E k &
 ./tcp6-socky -L -p 8002 -r -E k &
 ./tcp6-socky -L -p 8003 -r -E k &
 ./tcp6-socky -L -p 8004 -r -E k &
 ./tcp6-socky -L -p 8005 -r -E k &
 ./tcp6-socky -L -p 8006 -r -E k &

root@n5:/tmp# ./tcp6-socky -C -r -d 2001:db8:4::50 -p 8001 -X 8001
              -G cubic -E K 0 1400 13 &
              ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8002 -X 8002
              -G vegas -E K 0 1400 13 &
              ./tcp6-socky -C -r -d 2001:db8:4::50 -p 8003 -X 1113
              -G cubic -E K 0 1400 13 &
              ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8004 -X 3402
              -G vegas -E K 0 1400 13 &
              ./tcp6-socky -C -r -d 2001:db8:4::50 -p 8005 -X 8005
              -G cubic -E K 0 1400 13 &
              ./tcp6-socky -C -r -d 2001:db8:4::100 -p 8006 -X 8102
              -G vegas -E K 0 1400 13 &

proc 2934400 bytes in 13000392.000000 usec or 13.000392 sec
BW=1.505071 Mbps

proc 2445800 bytes in 13000321.000000 usec or 13.000321 sec
BW=1.805730 Mbps

proc 3166800 bytes in 13000397.000000 usec or 13.000397 sec
BW=1.948740 Mbps

proc 1920800 bytes in 13000287.000000 usec or 13.000287 sec
BW=1.182005 Mbps

proc 3355800 bytes in 13000206.000000 usec or 13.000206 sec
BW=2.065075 Mbps

proc 2938600 bytes in 13001011.000000 usec or 13.001011 sec
BW=1.808229 Mbps
```

Si se siguen agregando sesiones se puede ver que Vegas continua en la línea de perder el comportamiento activo y reacciona según lo indicado anteriormente, en este caso en la figura 8.18 se muestran 4 sesiones de cada grupo y se dibujan solo las de Vegas.

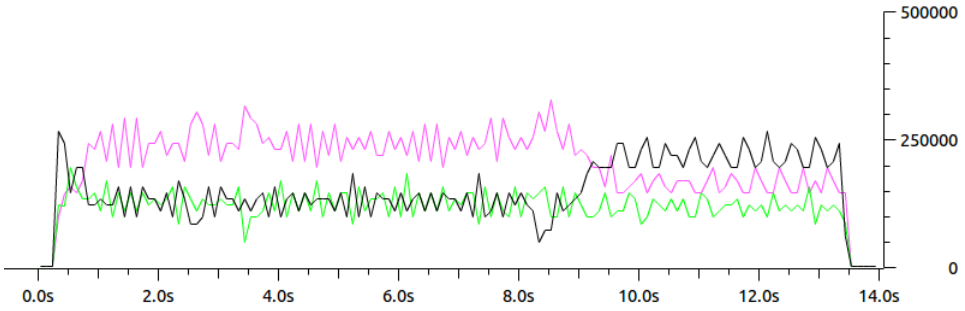


Figura 8.16: Tests combinados, 3 sesiones Cubic vs 3 Vegas, solo se grafican las sesiones Vegas

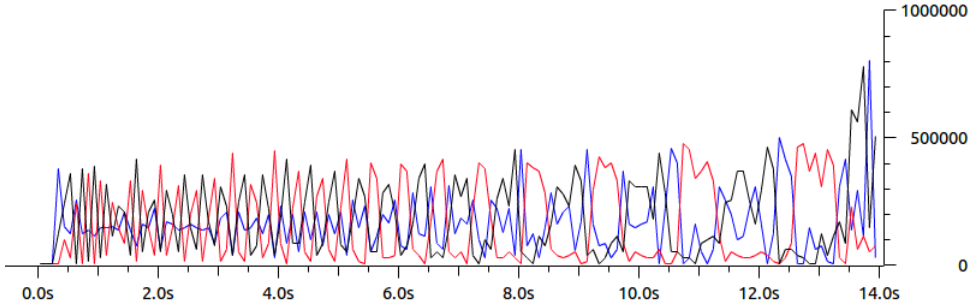


Figura 8.17: Tests combinados, 3 sesiones Cubic vs 3 Vegas, solo se grafican las sesiones Cubic

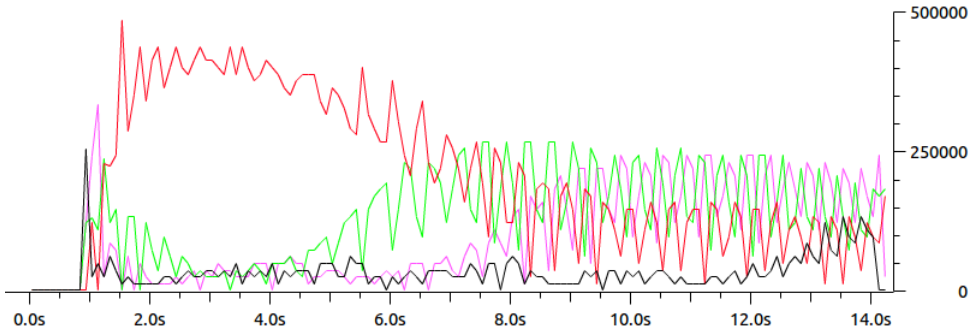


Figura 8.18: Tests combinados, 4 sesiones Cubic vs 4 Vegas, solo graficado Vegas

8.2.2. Conclusiones del segundo modelo

La utilización del Flow Label para poder diferenciar la forma en que reaccionarán con diferentes algoritmos de control de congestión los emisores de los flujos es posible de implementar y se muestra en práctica en el segundo modelo. En este caso se distingue entre los algoritmos reactivos y los pro-activos. Se muestra una implementación donde el modelo es de grano grueso y solo de distinguen los grupos dando la misma capacidad para cada uno de estos. Como ventaja tiene que es simple de implementar y los orígenes pueden marcar de forma pseudo-aleatoria el valor del campo, solo agregando al final del valor (en la parte menos significativa) una suma que permita distinguir en los 2 (dos) últimos bits (LSb) el tipo de control que está aplicando. El modelo es útil en ocasiones cuando el control de congestión es más conservador y se ve sobrepasado por CC tipo Reno, New-Reno o Cubic. De cualquier forma se ve en las pruebas que la implementación de Vegas al sumarse más de 2 flujos comienza a cambiar un tanto el comportamiento y parece tender a un algoritmo re-activo. Como desventaja se puede indicar que si no están equilibradas la cantidad de flujos o sesiones dentro de los dos grupos los que tengan menos sesiones obtendrán individualmente mayor ancho de banda digital. Otra alternativa sería dividir los recursos de forma equitativa entre todos los flujos y encolando por cada uno en clases diferentes, similar como se hace con FQ. El modelo tendría el problema que sería complejo de implementar en routers que se localizan en el núcleo de la red, debido a la cantidad de flujos que deberían procesar. En este caso la distinción de flujos debería hacerse usando la 3-tupla: dir. origen, dir. destino, flow label.
De acuerdo a lo estudiado en el capítulo anterior, la siguiente propuesta sigue más de cerca los lineamientos indicados por el IETF que la anterior. Los mismos se puntualizan a continuación.

- El modelo propuesto no requiere el mantenimiento de información a modo de estados o conocimiento previo.
- En el modelo propuesto permite usar los valores de los campos de la 3-tupla: dir. origen, dir. destino, Flow Label; para identificar que paquetes pertenecen a un flujo.
- Permite utilizan valores (pseudo)-aleatorios y solo se debe cambiar los 2 LSb para indicar el algoritmo de control de congestión.

Capítulo 9

Conclusiones

El documento ofrece los resultados del trabajo donde se investigan los modelos actuales de QoS en redes IP y principalmente muestra la posibilidad de extenderlos mediante el uso del campo **IPv6: Flow Label** (Etiqueta de flujo) en redes IPv6. Se realiza un análisis teórico y práctico de los modelos existentes.

Se investigan las diferentes propuestas de uso del campo apuntando a la QoS. Se desarrollan herramientas para mostrar la posibilidad de extensión de los modelos de QoS y se realizan numerosas pruebas en equipos sobre sistemas operativos reales.

Mediante la propuesta y desarrollo de 2 (dos) posibles nuevos modelos mostrados en el capítulo anterior se lleva a un ambiente real y práctico el manejo de QoS mediante IPv6. Se abren posibles usos y da como resultado que la utilización del campo para extender los modelos es útil.

En el primer modelo se muestra como cambia radicalmente la QoS ofrecida ante la presencia de competencia flujos si se atienden de acuerdo a la marca del Flow Label utilizando una asignación específica. En los tests donde se ignora el tratamiento, los resultados no son buenos, en cambio ante la asignación de las calidades se obtiene el tratamiento solicitado. Resultados similares se pueden obtener con un modelo DiffServ, pero el esquema presentado permite la solicitud y asignación de QoS mediante un campo de 20 bits, a diferencia de DiffServ que solo permitir asignar 6 bits del encabezado. Como contra de acuerdo a los requerimientos por RFCs del IETF, se crea un modelo con estados no recomendado. En el ejemplo presentado se requiere el mantenimiento de información a modo de estados o conocimiento previo, necesario para el tratamiento de los flujos de acuerdo a la marca y los routers dependen de este

valor para dar el tratamiento.

El segundo modelo está más acorde a los requerimientos IETF y apunta a diferenciar los algoritmos de control de congestión, la forma en que reaccionaran los generadores de los flujos, de acuerdo a valor de la marca en el Flow Label. En particular se presenta como se puede mejorar el uso de los algoritmos pro-activos, Vegas, sin que se vean afectados por los re-activos, Reno o Cubic por ejemplo, y permitir así que se pueda incrementar su utilización sobre las redes TCP/IP.

Como trabajos a futuro el documento presenta posibles extensiones: Una podría ser la combinación de las marcas de Flow Label para realizar balanceo de carga sobre enlaces asimétricos permitiendo diferentes QoS de acuerdo a los valores de ancho de banda u otros parámetros de los enlaces. También sería posible hacer un ruteo diferenciado a partir de estas marcas en los routers intermedios. Otro enfoque podría ser el uso del Flow Label para la identificación de flujos en un modelo de una red tipo SDN (Software Defined Network). Con respecto a la QoS las extensiones pueden apuntar a tratar de hacer el marcado de las etiquetas MPLS o la disponibilidad de QoS que tenga la capa de enlace en base a los valores de Flow Label IPv6.

Cerrando las conclusiones, se puede afirmar que con este trabajo se muestra que es posible usar el campo Flow Label de IPv6 para dar QoS a pesar de que la IETF sigue haciendo énfasis indicando que para QoS en IPv6 se debe aplicar DiffServ con el campo Traffic Class. El campo Flow Label hasta la actualidad sigue sin tener un uso específico masivo, pero parece tener potencial que se puede aprovechar desde el software para la gestión de redes en el plano de control, como es el caso actual de SDN. Este es un tema que constantemente esta evolucionando.

Apéndice A

Ejemplos de uso ICMPv6

A.1. ICMPv6, descubrimiento de vecinos

IPv6 remueve el protocolo ARP (RFC-826) de su stack y agrega la funcionalidad a ICMPv6. En IPv4, ICMP es obligatorio para que sea compatible con las RFCs, pero podría suprimerse y el protocolo seguirá funcionando. En cambio, en IPv6, ICMPv6 es fundamental. Los mensajes de ICMPv6 que reemplazan el ARP Request y ARP Reply son mensajes del protocolo **Network Discovery (ND)**, inicialmente definido en RFC-2461 y más tarde obsoleta por RFC-4861 [NNSS07]. Los mensajes para cumplir la función son los siguientes:

- ICMPv6 NS: Neighbor Solicitation (135).
- ICMPv6 NA: Neighbor Advertisment (136).

El mensaje ICMPv6 NS utiliza en la trama Ethernet una dirección destino multicast. La dirección comienza con `33:33:`, que es el prefijo que se define en RFC-3307 [Hab02] para IPv6 Neighbor Discovery. Los cuatro últimos bytes son mapeados desde la dirección destino IPv6. La dirección destino IPv6 del mensaje NS es una dirección IPv6 Multicast llamada **Solicited-node Multicast Address** que tienen el prefijo ff02::1:ff00:0/104 y el resto se completa con los últimos 24 bits de la dirección unicast IPv6. No se utiliza broadcast, aunque algunas primeras implementaciones no funcionando correctamente intentaban resolver el mapeo vía broadcast de L2. La respuesta, como es de esperar, es unicast. En el mensaje el valor del Next-Header para ICMPv6 es `0x3A(58)`.

Este proceso sólo se ejecuta para las direcciones unicast y no debe hacerse con las direcciones multicast. ICMPv6 también es utilizado en reemplazo de ARP para

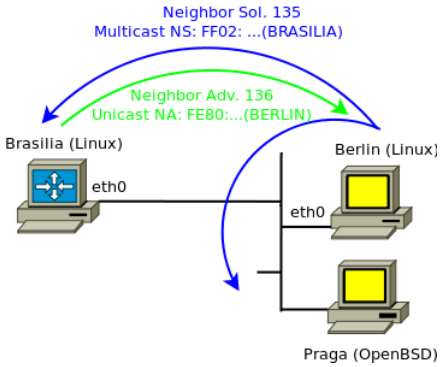


Figura A.1: Topología para el ejemplo de ND

mantener, no solo generar, la cache de mapeos de IPv6 a direcciones MAC-48. La cache se llama **Tabla/Cache de Vecinos**, que es el reemplazo de la **Cache ARP**.

El proceso de mantenimiento de las direcciones se llama **Neighbor Unreachability Detection (NUD)**. La confirmación de las entradas en la tabla de vecinos puede tener dos orígenes:

- Confirmación mediante las capas superiores.
- Mensajes ICMPv6 de NUD.

El proceso de NUD puede utilizar protocolos de capas superiores, como TCP o UDP, para determinar que un destino sigue estando accesible. Por ejemplo, si TCP recibe un ACK desde la dirección a renovar, esto sirve para refrescar la entrada. Si las capas superiores no pueden confirmar las direcciones en la cache, el ICMPv6 de forma activa tratará de validarlas mediante mensajes NUD explícitos. Se utilizan mensajes de ND llamados NUD. Estos mensajes son iguales a ICMPv6 NS y ICMPv6 NA, pero cambiando las direcciones destinos a unicast.

La topología del ejemplo ilustrativo se muestra en la imagen A.1.

Con el comando `ip(8)` en GNU/Linux, se inspecciona la tabla de vecinos. Esto reemplaza la tabla de ARP y al comando `arp(8)`.

```
root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3461 dev eth0 lladdr 52:54:00:12:34:61 nud reachable
```

De la misma forma que sucede con las entradas ARP, si estas se dejan de usar se eliminan después de un tiempo. **ICMPv6 NUD** mejora con respecto a ARP, en que,

valida las entradas en ambos sentidos, en lugar de hacerlo solo en uno, y optimiza los tiempos de estas consultas utilizando a las capas superiores (cuando es posible). Con **Upper Layer Positive Confirmation** las capas superiores deberían avisar cuando esperan una confirmación o un mensaje para validar las entradas en la cache.

Los estados de las entradas son los siguientes:

Failed: Fallado. Se ha borrado la entrada de forma forzada, manual o el intento de alcanzar el vecino ha sido fallido. Por ejemplo, no está activo el equipo.

Incomplete: Incompleto. el proceso de ND está por la mitad. Aún no se tiene respuesta del vecino.

Reachable: Alcanzable. En la tabla de vecinos se tiene la entrada como válida. Se envió un NS y se recibió un NA para una entrada en particular o se ha confirmado desde las capas superiores

Stale: Caduco. El tiempo ha expirado, pero el vecino no ha sido detectado como inalcanzable aún por NUD. Queda en este estado hasta que se envíe un mensaje al vecino.

Delay: Retrasado. la entrada estaba Stale, pero se ha enviado un mensaje a la última dirección conocida esperando que las capas superiores confirmen la entrada. Si no se confirma, deberá enviarse un NS.

Probe: En proceso. se ha enviado un NS unicast y aún no se recibe respuesta.

Permanent: Permanente. se ha agregado la entrada en la tabla de forma estática. No tiene timer de expiración asociado.

Además, los mensajes ICMPv6 tienen varios flags:

R(Router flag): se configura cuando el mensaje es enviado por un router. El R-bit se usa para NUD cuando un router pasa a ser host.

S(Solicited flag): se configura el S-bit para indicar que el aviso, NA, va en respuesta a una solicitud NS. No se debe colocar en mulitcast NA ni en NA no solicitados.

O(Override flag): se configura para indicar que el NA debe usarse para actualizar las entradas en la cache de link-layer. No debería ser usado para NA no solicitados. Si para aquellos no solicitados.

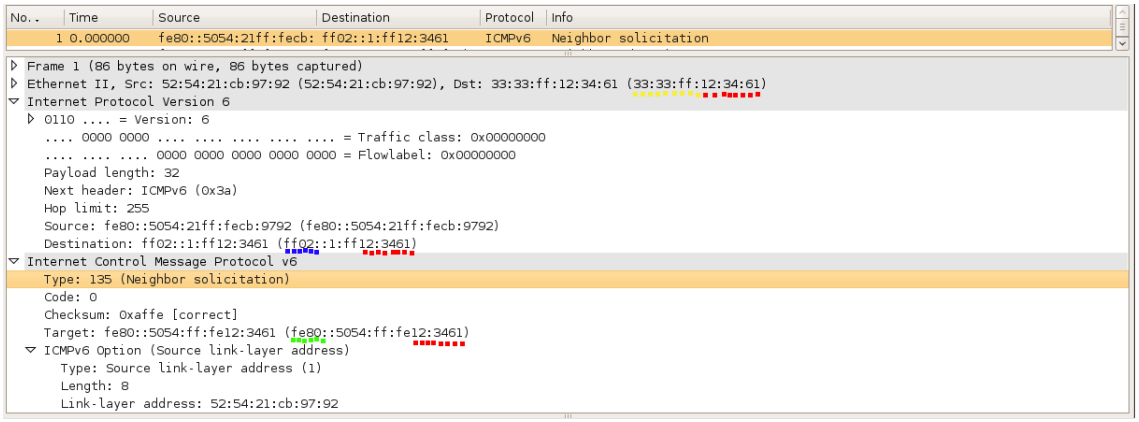


Figura A.2: Captura del mensaje de Neighbor Solicitation

En el ejemplo se ve como inicialmente se suprime una entrada particular y como consultar el estado de la tabla de vecinos.

```
root@berlin:~# ip -f inet6 neigh del fe80::5054:ff:fe12:3461 dev eth0

root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3461 dev eth0  router nud failed
```

Luego intentar alcanzar al vecino, por la dirección global o la de Link-Local.

```
root@berlin:~# ping6 -c 2 -I eth0 fe80::5054:ff:fe12:3461
PING fe80::5054:ff:fe12:3461(fe80::5054:ff:fe12:3461) from
     fe80::5054:21ff:fecb:9792 eth0: 56 data bytes
64 bytes from fe80::5054:ff:fe12:3461: icmp_seq=1 ttl=64 time=5.76 ms
64 bytes from fe80::5054:ff:fe12:3461: icmp_seq=2 ttl=64 time=1.62 ms

--- fe80::5054:ff:fe12:3461 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.622/3.692/5.762/2.070 ms
```

Las entradas ahora están en las tablas de los dos equipos (origen y destino).

```
root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3461 dev eth0 lladdr 52:54:00:12:34:61 nud reachable

root@brasilia:~# ip -f inet6 neigh show
fe80::5054:21ff:fecb:9792 dev eth0 lladdr 52:54:21:cb:97:92 nud reachable
```

Si observamos el contenido de los paquetes capturados se encuentran los dos mensajes NS y NA que se observan en las imágenes de las capturas A.2 y A.3 respectivamente.

Si se vuelve a lanzar una prueba antes que expire el timer y se marquen las entradas como Stale, se puede ver que no se usa el proceso ND, ya que la confirman las capas superiores.

Si no hay tráfico, después de un tiempo, el timer asociado expira y se marcara como Stale. Inmediatamente, al volver a realizar la prueba se marcará como Delay,

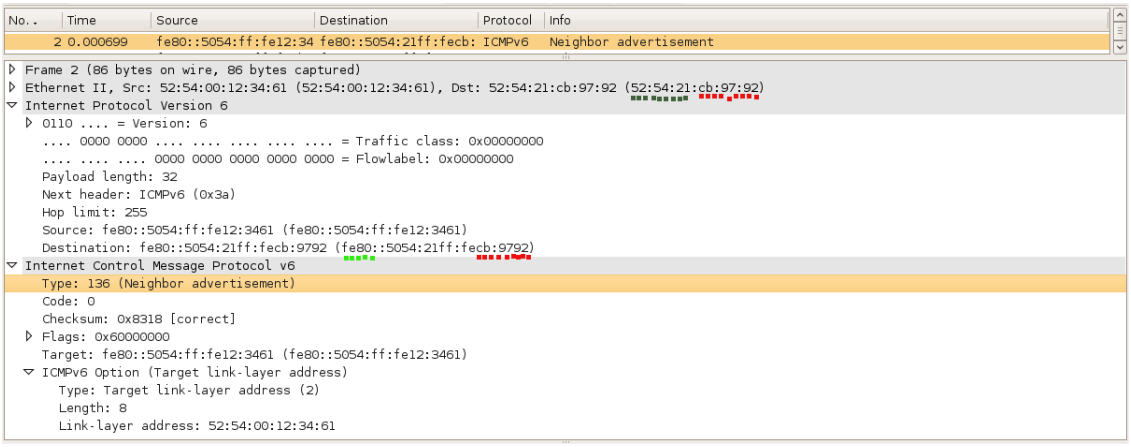


Figura A.3: Captura del mensaje de Neighbor Advertisement

ya que se espera que las capas superiores que la utilizan la confirmen, en este caso ICMPv6 (ECHO Request).

```
root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3461 dev eth0 lladdr 52:54:00:12:34:61 nud stale

root@berlin:~# ping6 -c 2 -I eth0 fe80::5054:ff:fe12:3461
PING fe80::5054:ff:fe12:3461(fe80::5054:ff:fe12:3461)
     from fe80::5054:21ff:fecb:9792 eth0: 56 data bytes
64 bytes from fe80::5054:ff:fe12:3461: icmp_seq=1 ttl=64 time=2.37 ms
64 bytes from fe80::5054:ff:fe12:3461: icmp_seq=2 ttl=64 time=1.67 ms

--- fe80::5054:ff:fe12:3461 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.671/2.020/2.370/0.352 ms

root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3461 dev eth0 lladdr 52:54:00:12:34:61 nud delay
```

Si se confirma, la entrada vuelve al estado Reachable (Alcanzable). Si la capa superior no la confirma, se debe enviar un NS y, al recibir el NA correspondiente, se la debe marcar como Reachable. Al final de la captura se ve el proceso de NUD vía ICMPv6.

```
root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3461 dev eth0 lladdr 52:54:00:12:34:61 nud reachable
```

Si deja de existir actividad para una dirección dada, esta se dará de baja estando en Reachable, pasando luego por el estado Stale, y, finalmente, borrada de la tabla. En el caso que se requiera, también, se pueden agregar las entradas de forma estática.

```
root@berlin:~# ip -f inet6 neigh add fe80::5054:ff:fe12:3462
               lladdr 52:54:00:12:34:61 dev eth0

root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3462 dev eth0 lladdr 52:54:00:12:34:61 nud permanent
fe80::5054:ff:fe12:3461 dev eth0 lladdr 52:54:00:12:34:61 nud stale
```

De esta forma, no sería necesario usar el proceso ND. En el ejemplo se podrían ver mensajes NS multicast, pero serían del otro extremo ya que este no tiene una entrada estática para responder y posiblemente la entrada ha sido eliminada de la tabla. Si no es alcanzable, por ejemplo, intentado acceder a un equipo no existente quedará como Failed.

```
root@berlin:~#  ping6 -c 2 -I eth0 fe80::5054:ff:fe12:3464 &
PING fe80::5054:ff:fe12:3464(fe80::5054:ff:fe12:3464)
from fe80::5054:21ff:fecb:9792 eth0: 56 data bytes

root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3464 dev eth0  nud incomplete

From fe80::5054:21ff:fecb:9792 icmp_seq=1 Destination unreachable:
                                          Address unreachable
From fe80::5054:21ff:fecb:9792 icmp_seq=2 Destination unreachable:
                                          Address unreachable

--- fe80::5054:ff:fe12:3464 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1000ms

root@berlin:~# ip -f inet6 neigh show
fe80::5054:ff:fe12:3464 dev eth0  nud failed
```

A.2. ICMPv6, proceso de auto-configuración sin estado

El proceso de auto-configuración se puede aplicar a direcciones locales o globales. Para el caso de link-local, el proceso es independiente y no requiere más información que la aportada por el equipo, aunque genera un proceso de DAD (Duplicate Address Detection) para ver que no exista un problema de configuración generando direcciones duplicadas. Si se quiere tener una dirección global que permita acceder a equipos de otras redes se debe usar un prefijo global. En un caso, en el router se podría configurar manualmente.

A.2.1. Configuración manual/estática

Ejemplo de configuración manual.

```
root@brasilia:~# ifconfig eth0 inet6 add 2001:0DB8:1234:0001::1/64
root@brasilia:~# ifconfig eth1 inet6 add 2001:0DB8:1234:0000::254/64

root@brasilia:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:61
          inet addr:172.20.1.5  Bcast:172.20.1.255  Mask:255.255.255.0
          inet6 addr: 2001:db8:1234:1::1/64 Scope:Global
          inet6 addr: fe80::5054:ff:fe12:3461/64 Scope:Link
          ...

root@brasilia:~# netstat -nr -6
Kernel IPv6 routing table
Destination                 Next Hop   Flags Metric Ref    Use Iface
```

```
::1/128                     ::         U     0      2        2 lo
2001:db8:1234::254/128      ::         U     0      0        2 lo
2001:db8:1234::/64          ::         U     256    0        0 eth1
2001:db8:1234:1::1/128      ::         U     0      0        2 lo
2001:db8:1234:1::/64        ::         U     256    0        0 eth0
fe80::5054:ff:fe12:3461/128 ::         U     0      9        2 lo
fe80::5054:ff:fe12:3462/128 ::         U     0      0        2 lo
fe80::/64                   ::         U     256    0        0 eth0
fe80::/64                   ::         U     256    0        0 eth1
ff00::/8                    ::         U     256    0        0 eth0
ff00::/8                    ::         U     256    0        0 eth1
```

La configuración se puede realizar con los comandos `ip(8)`, obteniendo los mismos resultados.

```
root@brasilia:~# ip addr add 2001:0DB8:1234:0001::1/64   dev eth0
root@brasilia:~# ip addr add 2001:0DB8:1234:0000::254/64 dev eth1

root@brasilia:~# ip addr show dev eth1
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 52:54:00:12:34:62 brd ff:ff:ff:ff:ff:ff
    inet 172.20.0.254/24 brd 172.20.0.255 scope global eth1
    inet6 2001:db8:1234::254/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe12:3462/64 scope link
       valid_lft forever preferred_lft forever

root@brasilia:~# ip -f inet6 route show
2001:db8:1234::/64 dev eth1  metric 256  mtu 1500 advmss 1440 hoplimit
                                                          4294967295
2001:db8:1234:1::/64 dev eth0  metric 256  mtu 1500 advmss 1440 hoplimit
                                                          4294967295
fe80::/64 dev eth0  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
fe80::/64 dev eth1  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
ff00::/8 dev eth0  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
ff00::/8 dev eth1  metric 256  mtu 1500 advmss 1440 hoplimit 4294967295
unreachable default dev lo  proto none  metric -1  error -101 hoplimit
                                                          255
```

Para el caso de los hosts se puede también recurrir a la configuración manual.

```
root@berlin:~# ifconfig eth0 inet6 add 2001:0DB8:1234:0001::100/64
root@berlin:~# ip route add 2001:db8:1234::/64 via 2001:db8:1234:1::1

root@berlin:~# ping6 -c 2 2001:db8:1234::254
PING 2001:db8:1234::254(2001:db8:1234::254) 56 data bytes
64 bytes from 2001:db8:1234::254: icmp_seq=1 ttl=64 time=1.75 ms
64 bytes from 2001:db8:1234::254: icmp_seq=2 ttl=64 time=1.16 ms
...
```

O se puede utilizar métodos de configuración automáticos con estados como DHCPv6 o sin estados como SAC mediante RADV (Router Advertisement) que se verá a continuación. El proceso de SAC estuvo definido en RFC-2462 y fue reemplazado por RFC-4862 [TNJ07].

El proceso de configuración de direcciones estáticas no es el recomendado para los hosts ya que al ser direcciones muy largas es posible cometer errores, además de

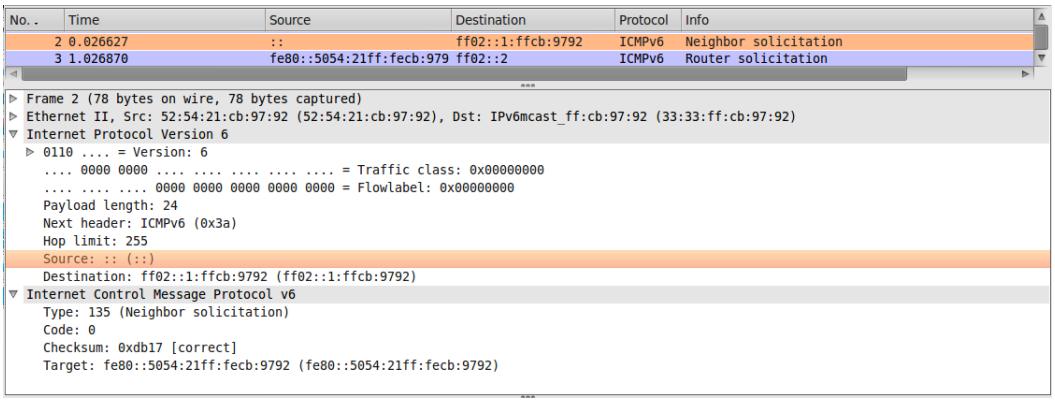


Figura A.4: Captura de un mensaje de DAD

lo tedioso que puede resultar escribirlas. La dirección link-local auto-configurada no sirve porque no es una dirección ruteable. Por estas razones, IPv6 define dos métodos para poder asignar direcciones ruteables de forma auto-configurada, uno Stateless conocido como **Stateless Address Autoconfiguration (SAC)** y otro Stateful que utiliza DHCPv6, una extensión del DHCP.

A.2.2. ICMPv6 Duplicate Address Detection y Router Discovery

Cada vez que se inicia o configura una interfaz que tiene habilitado el soporte de IPv6, se genera un mensaje ICMPv6 NS para detectar si ya existe configurada una interfaz, en la red local, con la misma dirección. Si nadie responde, se supone que la dirección es única. El proceso se conoce como **Duplicate Address Detection (DAD)** y se debe ejecutar por cada una de las direcciones unicast asignadas a la interfaz. Como la dirección que se está comprobando su unicidad en el enlace no puede ser utilizada hasta terminar exitosamente el proceso DAD se utiliza como dirección origen del mensaje la Unspecified Address ::. La dirección destino del mensaje IPv6 es una dirección de multicast y dentro del ICMPv6 va la dirección unicast de la cual se está testeando su unicidad. La figura de la captura correspondiente a un DAD de Link-Local es la A.4.

Podría suceder que la dirección testeada esté asignada a otro nodo. En ese caso se debería recibir una respuesta NA indicando que la dirección está duplicada. Si el proceso tiene éxito, la dirección se asigna a la interfaz y, como siguiente paso, se intenta descubrir la presencia de un router en el enlace.

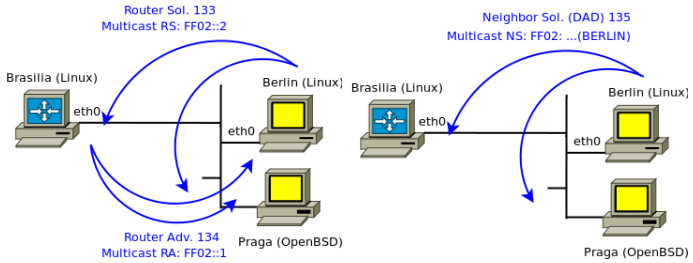


Figura A.5: Ejemplo de RS/RA y DAD

Para poder auto-configurar sus direcciones (otras que no sean la de link-layer), los hosts puede intentar encontrar un router que les provea la información que les falta para realizar ese trabajo. El proceso se llama **Router Discovery (RD)** y está compuesto por los siguientes mensajes:

- ICMPv6 Router Solicitation RS (133).
- ICMPv6 Router Advertisment RA (134).

Los hosts, cuando habilitan IPv6 en las interfaces, envían un mensaje ICMPv6, **Router Solicitation**, para descubrir la presencia de un router vecino. Si existe un router, le contestará con un **Router Advertisement**. En el mensaje, el router puede indicar uno, o varios, prefijos de 64 bits, que junto con el identificador de interfaz de 64 bits propio de la interfaz son utilizados por los nodos para formar una dirección IPv6 Stateless. El proceso se conoce como **Stateless Address Autoconfiguration**. El alcance de las direcciones generadas depende del alcance de los prefijos recibidos del router. Si el router lo permite, los hosts pueden establecer al router como Default Gateway. El diagrama del ejemplo de Router Solicitation/Advertisement y DAD es mostrada en las figura A.5.

Además, los routers envían, periódicamente, **Unsolicited Router Advertisements (URA)** para anunciar cambios y ayudar a la re-auto-configuración de los hosts. Los routers anuncian vía los RA, el/los prefijo/s de la red y su propia dirección. Los hosts deben continuar escuchando por cambios de prefijos. Los routers no obtienen sus configuraciones de forma dinámica, deben ser configurados manualmente. A continuación, se muestra un ejemplo. Primero, en el router es necesario configurar manualmente las direcciones y el Router Advertisment Server. En el caso de GNU/Linux es un daemon (proceso de sistema) que se debe instalar.

```
root@brasilia:~# vi /etc/radvd.conf
root@brasilia:~# cat /etc/radvd.conf
interface eth0

    AdvSendAdvert on;
    prefix 2001:db8:1234:1::/64  ;
;
```

Como el equipo se va a comportar como un router se deberá habilitar el forwarding. El script de inicio del proceso `radvd(8)` lo hace. También se puede hacer manualmente.

```
root@brasilia:~# sysctl net.ipv6.conf.all.forwarding=1

root@brasilia:~# sysctl net.ipv6 | grep forward
net.ipv6.conf.default.forwarding = 1
net.ipv6.conf.all.forwarding = 1
net.ipv6.conf.eth1.forwarding = 1
net.ipv6.conf.eth0.forwarding = 1
net.ipv6.conf.lo.forwarding = 1

root@brasilia:~# /etc/init.d/radvd start
Starting radvd: radvd.

root@brasilia:~# ps -elf | grep radvd
1 S radvd 5079  1 0 84 0 - 447 - 19:16 ? 00:00:00 /usr/sbin/radvd -u radvd..
```

La configuración manual de las direcciones se mostró anteriormente. Los mensajes de RA son lanzados al iniciarse el proceso y, posteriormente, con una frecuencia determinada. Esta se puede configurar en el archivo del daemon con el parámetro:

```
MaxRtrAdvInterval seconds # Min 4 sec, Max 1800 sec, Default 600 seconds
```

Los hosts pueden configurarse para aceptarlos o no. Por ejemplo, en GNU/Linux.

```
root@berlin:~# sysctl net.ipv6.conf.eth0.accept_ra
net.ipv6.conf.eth0.accept_ra = 1
```

También se pueden configurar los siguientes parámetros:

```
## Delay para enviar un RS despues que la interfaz está UP
root@berlin:~# sysctl net.ipv6.conf.eth0.router_solicitation_delay
net.ipv6.conf.eth0.router_solicitation_delay = 1

## Segundos a esperar entre cada RS enviado
root@berlin:~# sysctl net.ipv6.conf.eth0.router_solicitation_interval
net.ipv6.conf.eth0.router_solicitation_interval = 4

## Cantidad de RS enviados hasta que se asume que no hay Router
root@berlin:~# sysctl net.ipv6.conf.eth0.router_solicitations
net.ipv6.conf.eth0.router_solicitations = 3
```

Para poder ver los RS al inicio se puede hace un down(desactivar) y up(activar) de la interfaz en el cliente/host Berlin. Para el caso, aún no está habilitado el RADVD, por eso se envían 3 RS y no se obtiene ninguna respuesta. Si se lo realiza nuevamente,

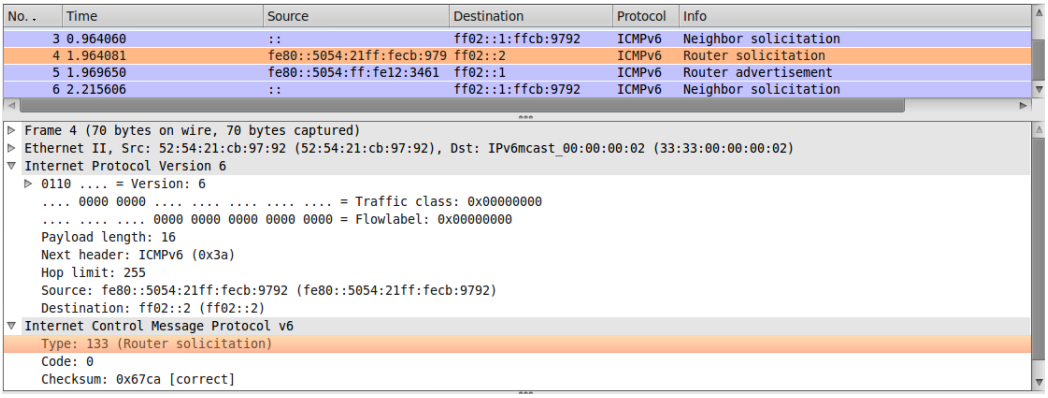


Figura A.6: Captura de un mensaje de RS

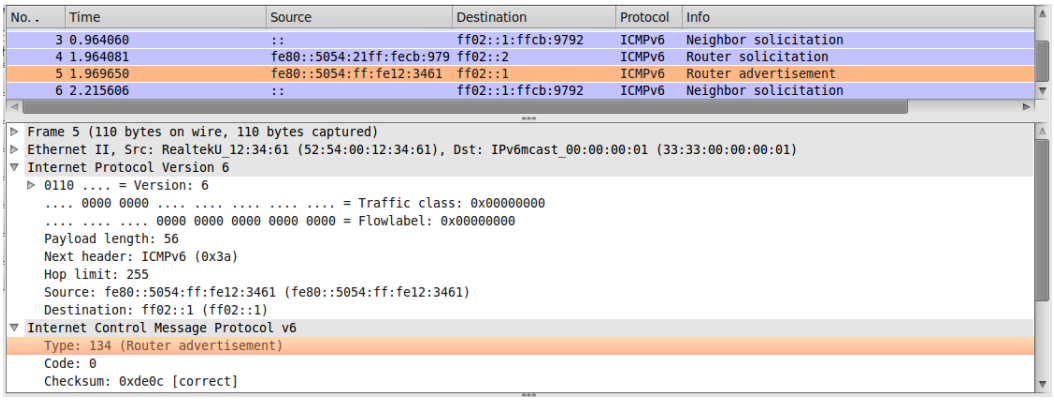


Figura A.7: Captura de un mensaje de RA

una vez activado el daemon, ahora se ve la respuesta al RS. Como resultado se obtiene una dirección IPv6 global de forma automática. Las imágenes de las capturas del RS y el RA son A.6 y A.7 en este orden.

Finalmente se puede ver el prefijo y como se genera la dirección global.

```
root@berlin:~# ifconfig eth0 | grep Global
          inet6 addr: 2001:db8:1234:1::100/64 Scope:Global
          inet6 addr: 2001:db8:1234:1:5054:21ff:fecb:9792/64 Scope:Global
```

En la prueba mostrada, el proceso que se usa para generar los últimos 64 es el mismo que para las direcciones link-local, basándose en el EUI-64.

```
eth0      Link encap:Ethernet  HWaddr 52:54:21:CB:97:92
          ...
          inet6 addr: fe80::5054:21ff:fecb:9792/64 Scope:Link
          ...
          inet6 addr: 2001:db8:1234:1:5054:21ff:fecb:9792/64 Scope:Global
```

Se puede observar que la entrada, en la tabla de vecinos se agrega un atributo que identifica al mismo como router. Ahora, los clientes nunca van a borrar de la tabla de vecinos a su default gateway porque la entrada se refrescará con los RA. La dirección que se agrega es la de link-local y no la global del router para el segmento de red.

```
root@berlin:~# ip -f inet6 neigh
fe80::5054:ff:fe12:3461 dev eth0 lladdr 52:54:00:12:34:61 router nud stale
```

Con esta configuración se genera la dirección global en base al valor derivado de la dirección MAC, IEEE ID. No necesariamente todos los nodos tienen interfaces con direcciones físicas/MAC con IEEE IDs, como puede ser el caso de un enlace PPP WAN. En esta situación podría recurrir a la dirección MAC de otra interfaz del equipo. Podría suceder no tener ningún recurso para generar dicho valor. Otra propuesta podría ser utilizar valores (pseudo)-aleatorios, aunque podría resultar en que no sean únicos dentro del segmento. Las configuraciones que mantienen el IEEE ID en la dirección global son cuestionadas y se proponen nuevos mecanismos a partir de combinar algoritmos para generar valores (pseudo)-aleatorios a partir de los identificadores IEEE para obtener valores únicos. El RFC RFC-4941 [NDK07], previamente definido en RFC-3041 trata el tema. El documento brinda extensiones de Privacidad para la configuración SAC, generando valores que esconden el IEEE ID e incluso que varían, de acuerdo al tiempo de vida de la dirección IPv6. Por ejemplo, si se le configura que genere direcciones temporales, usará un algoritmo que ofusca la dirección MAC en la generación.

```
root@berlin:~# echo 1 > /proc/sys/net/ipv6/conf/eth0/use_tempaddr

root@berlin:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 52:54:00:12:34:56
          inet addr:172.20.1.100  Bcast:172.20.1.255  Mask:255.255.255.0
          inet6 addr: 2001:db8:1234:1::100/64 Scope:Global
          inet6 addr: 2001:db8:1234:1:6a8f:b83:498b:e4e0/64 Scope:Global
          inet6 addr: 2001:db8:1234:1:5054:ff:fe12:3456/64 Scope:Global
          inet6 addr: fe80::5054:ff:fe12:3456/64 Scope:Link
          ...
```

El parámetro en GNU/Linux puede tener los siguientes valores: 0:no usar extensiones de privacidad, 1: usarlas pero preferir públicas sobre temporales, 2: prefiere temporales.

```
root@berlin:~# ping6 2001:db8:1234:1::4
PING 2001:db8:1234:1::1(2001:db8:1234:1::4) 56 data bytes
From 2001:db8:1234:1:5054:ff:fe12:3456 icmp_seq=2 Destination unreachable:
                                                  Address unreachable
From 2001:db8:1234:1:5054:ff:fe12:3456 icmp_seq=3 Destination unreachable:
                                                  Address unreachable
From 2001:db8:1234:1:5054:ff:fe12:3456 icmp_seq=4 Destination unreachable:
                                                  Address unreachable

--- 2001:db8:1234:1::4 ping statistics ---
5 packets transmitted, 0 received, +3 errors, 100% packet loss, time 3999ms

root@berlin:~# echo 2 > /proc/sys/net/ipv6/conf/eth0/use_tempaddr
root@berlin:~# ping6 2001:db8:1234:1::4
PING 2001:db8:1234:1::1(2001:db8:1234:1::4) 56 data bytes
From 2001:db8:1234:1:6a8f:b83:498b:e4e0 icmp_seq=2 Destination unreachable:
                                                    Address unreachable
From 2001:db8:1234:1:6a8f:b83:498b:e4e0 icmp_seq=3 Destination unreachable:
                                                    Address unreachable
From 2001:db8:1234:1:6a8f:b83:498b:e4e0 icmp_seq=4 Destination unreachable:
                                                    Address unreachable

--- 2001:db8:1234:1::4 ping statistics ---
4 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2999ms
```

En el mensaje Router Advertisment se puede anunciar otros parámetros, por ejemplo, el MTU. El MTU anunciado no debe ser menor a 1280 bytes en IPv6. En este caso, los enlaces tiene el valor de Ethernet. Recientemente también se permiten agregar rutas más específicas o un parámetro que era necesario para una configuración completa (*full-configuration*) de host, los DNS servers, primero por RFC-5008 y luego reemplazada por RFC-6106 [JPBM10].

```
root@brasilia:~# cat /etc/radvd.conf

interface eth0

    AdvSendAdvert on;
    AdvLinkMTU 1280;
    prefix 2001:db8:1234:1::/64  ;

#
# RDNSS
# NOTA: No muy difundida funcionalidad
#
        RDNSS 2001:db8::1 2001:db8::2

                AdvRDNSSLifetime 30;
        ;

#
# Lista de nombres de dominios
#
        DNSSL info.unlp.test unlp.test

                AdvDNSSLLifetime 30;
        ;
;

root@brasilia:~# /etc/init.d/radvd restart
```

En caso de no contar con esta funcionalidad, los servidores de DNS se podrán

obtener por el proceso de DHCPv6 sin necesidad que conserve estados, ya que solo entregará direcciones IPv6 de servidores fijos.

Apéndice B

Ejemplo con IntServ y RSVP

En este apéndice se construirá un escenario de ejemplo donde se muestra RSVP en forma práctica. En las pruebas, RSVP trabajará con las técnicas de encolado de los routers, por ejemplo, WFQ o RED. Como primer paso en una configuración de RSVP se debe planificar y evaluar los recursos disponibles.

En general los equipos que implementan RSVP no permiten usar el 100 % de los recursos, por ejemplo, un 75 % suele ser el valor por default que el protocolo tendrá para administrar. También se puede limitar la cantidad de ancho de banda digital que puede reservar un solo flujo. El porcentaje restante es para el funcionamiento de los protocolos de control, como el ruteo y el mismo RSVP.

En la topología del ejemplo se usan 4 routers conectados formando un cuadrilátero como la figura B.1 basada en equipos cisco. Los pasos iniciales son configurar la red IPv4 y seleccionar el protocolo de ruteo, en este caso OSPF. Esto se hace en todos los routers del ejemplo.

```
2001(config)#
!
hostname 2001
!
interface Loopback0
 ip address 1.0.0.1 255.255.255.255
!
interface Serial2/0
 description ####_2002.S2/0_####
 ip address 10.0.0.1 255.255.255.0
 serial restart-delay 0
 no shut
!
interface Serial2/1
 description ####_2003.Serial3/0_####
 ip address 13.0.0.1 255.255.255.0
 serial restart-delay 0
 no shut
!
```

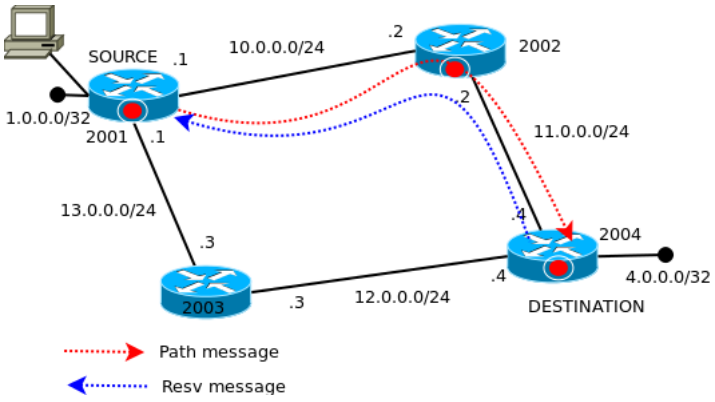


Figura B.1: Topología de prueba para ver RSVP trabajando

```
router ospf 1
 log-adjacency-changes
 network 1.0.0.0 0.0.0.255 area 0
 network 10.0.0.0 0.0.0.255 area 0
 network 13.0.0.0 0.0.0.255 area 0
!
end
```

```
2002(config)#
!
hostname 2002
!
interface Loopback0
 ip address 2.0.0.1 255.255.255.255
!
interface Serial2/0
 description ####_2001.Serial2/0_####
 ip address 10.0.0.2 255.255.255.0
 serial restart-delay 0
 no shut
!
interface Serial2/1
 description ####_2004.Serial2/1_####
 ip address 11.0.0.2 255.255.255.0
 serial restart-delay 0
 no shut
!
router ospf 1
 log-adjacency-changes
 network 10.0.0.0 0.0.0.255 area 0
 network 11.0.0.0 0.0.0.255 area 0
!
end
```

```
2003(config)#
!
hostname 2003
!
interface Loopback0
 ip address 3.0.0.1 255.255.255.255
!
interface Serial3/0
 description ####_2001.S2/1_####
 ip address 13.0.0.3 255.255.255.0
```

```
 serial restart-delay 0
 no shut
!
interface FastEthernet1/0
 description ####_2004.F1/0_####
 ip address 12.0.0.3 255.255.255.0
 duplex auto
 speed auto
 no shut
!
interface Ethernet2/0
 description ####_Host_####
 mac-address 2222.2222.2222
 ip address 20.0.0.3 255.255.255.0
 duplex half
 no shut
!
router ospf 1
 log-adjacency-changes
 network 13.0.0.0 0.0.0.255 area 0
 network 20.0.0.0 0.0.0.255 area 0
!
end
```

```
2004(config)#
!
hostname 2004
!
interface Loopback0
 ip address 4.0.0.1 255.255.255.255
!

interface FastEthernet1/0
 description ####_2003.F1/0_####
 ip address 12.0.0.4 255.255.255.0
 no shut
 duplex auto
 speed auto
!
interface Serial2/1
 description ####_2002.S2/1_####
 ip address 11.0.0.4 255.255.255.0
 serial restart-delay 0
 no shut
!
router ospf 1
 log-adjacency-changes
 network 4.0.0.0 0.0.0.255 area 0
 network 11.0.0.0 0.0.0.255 area 0
 network 12.0.0.0 0.0.0.255 area 0
!
end
```

Luego, ver el funcionamiento del protocolo de ruteo e inspeccionar la RIB, tabla de ruteo, en cada nodo.

```
2001#show ip ospf neighbor

Neighbor ID     Pri   State      Dead Time   Address         Interface
3.0.0.1           0   FULL/  -   00:00:38    13.0.0.3        Serial2/1
2.0.0.0           0   FULL/  -   00:00:33    10.0.0.2        Serial2/0

2001#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
```

```
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       L2 - IS-IS level-2 ia - IS-IS inter area, * - candidate default,
       U - per-user static route o - ODR,
       P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
C       1.0.0.1 is directly connected, Loopback0
     4.0.0.0/32 is subnetted, 1 subnets
O       4.0.0.1 [110/129] via 10.0.0.2, 00:16:33, Serial2/0
     20.0.0.0/24 is subnetted, 1 subnets
O       20.0.0.0 [110/74] via 13.0.0.3, 00:09:51, Serial2/1
     10.0.0.0/24 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, Serial2/0
     11.0.0.0/24 is subnetted, 1 subnets
O       11.0.0.0 [110/128] via 10.0.0.2, 00:16:33, Serial2/0
     12.0.0.0/24 is subnetted, 1 subnets
O       12.0.0.0 [110/129] via 10.0.0.2, 00:04:15, Serial2/0
     13.0.0.0/24 is subnetted, 1 subnets
C       13.0.0.0 is directly connected, Serial2/1
```

```
2002#show ip ospf neighbor

Neighbor ID     Pri   State      Dead Time   Address         Interface
4.0.0.1           0   FULL/  -   00:00:37    11.0.0.4        Serial2/1
1.0.0.1           0   FULL/  -   00:00:38    10.0.0.1        Serial2/0

2002#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       L2 - IS-IS level-2 ia - IS-IS inter area, * - candidate default,
       U - per-user static route o - ODR,
       P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
O       1.0.0.1 [110/65] via 10.0.0.1, 00:16:41, Serial2/0
     2.0.0.0/32 is subnetted, 1 subnets
C       2.0.0.1 is directly connected, Loopback0
     4.0.0.0/32 is subnetted, 1 subnets
O       4.0.0.1 [110/65] via 11.0.0.4, 00:16:41, Serial2/1
     20.0.0.0/24 is subnetted, 1 subnets
O       20.0.0.0 [110/138] via 10.0.0.1, 00:10:07, Serial2/0
     10.0.0.0/24 is subnetted, 1 subnets
C       10.0.0.0 is directly connected, Serial2/0
     11.0.0.0/24 is subnetted, 1 subnets
C       11.0.0.0 is directly connected, Serial2/1
     12.0.0.0/24 is subnetted, 1 subnets
O       12.0.0.0 [110/65] via 11.0.0.4, 00:04:33, Serial2/1
     13.0.0.0/24 is subnetted, 1 subnets
O       13.0.0.0 [110/128] via 10.0.0.1, 00:13:57, Serial2/0
```

```
2003#show ip ospf neighbor

Neighbor ID     Pri   State      Dead Time   Address         Interface
1.0.0.1           0   FULL/  -   00:00:32    13.0.0.1        Serial3/0
```

```
2003#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       L2 - IS-IS level-2 ia - IS-IS inter area, * - candidate default,
       U - per-user static route o - ODR,
       P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
O       1.0.0.1 [110/65] via 13.0.0.1, 00:10:14, Serial3/0
     3.0.0.0/32 is subnetted, 1 subnets
C       3.0.0.1 is directly connected, Loopback0
     4.0.0.0/32 is subnetted, 1 subnets
O       4.0.0.1 [110/193] via 13.0.0.1, 00:10:14, Serial3/0
     20.0.0.0/24 is subnetted, 1 subnets
C       20.0.0.0 is directly connected, Ethernet2/0
     10.0.0.0/24 is subnetted, 1 subnets
O       10.0.0.0 [110/128] via 13.0.0.1, 00:10:14, Serial3/0
     11.0.0.0/24 is subnetted, 1 subnets
O       11.0.0.0 [110/192] via 13.0.0.1, 00:10:14, Serial3/0
     12.0.0.0/24 is subnetted, 1 subnets
C       12.0.0.0 is directly connected, FastEthernet1/0
     13.0.0.0/24 is subnetted, 1 subnets
C       13.0.0.0 is directly connected, Serial3/0
```

```
2004#show ip ospf neighbor

Neighbor ID     Pri   State     Dead Time   Address         Interface
2.0.0.0           0   FULL/  -  00:00:30    11.0.0.2        Serial2/1

2004#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
       L2 - IS-IS level-2 ia - IS-IS inter area, * - candidate default,
       U - per-user static route o - ODR,
       P - periodic downloaded static route

Gateway of last resort is not set

     1.0.0.0/32 is subnetted, 1 subnets
O       1.0.0.1 [110/129] via 11.0.0.2, 00:16:45, Serial2/1
     4.0.0.0/32 is subnetted, 1 subnets
C       4.0.0.1 is directly connected, Loopback0
     20.0.0.0/24 is subnetted, 1 subnets
O       20.0.0.0 [110/202] via 11.0.0.2, 00:10:22, Serial2/1
     10.0.0.0/24 is subnetted, 1 subnets
O       10.0.0.0 [110/128] via 11.0.0.2, 00:16:45, Serial2/1
     11.0.0.0/24 is subnetted, 1 subnets
C       11.0.0.0 is directly connected, Serial2/1
     12.0.0.0/24 is subnetted, 1 subnets
C       12.0.0.0 is directly connected, FastEthernet1/0
     13.0.0.0/24 is subnetted, 1 subnets
O       13.0.0.0 [110/192] via 11.0.0.2, 00:15:34, Serial2/1
```

El protocolo de ruteo fue configurado para generar un camino lineal de forma de que los mensajes desde el router con ID 2004 hacia 2003 deban pasar primero por 2001, luego por 2002 para finalmente llegar a 2004. Haciendo unas pruebas desde 2003 a 2004 se puede ver.

```
2003#ping 4.0.0.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 4.0.0.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 12/18/28 ms

2003#traceroute 4.0.0.1 numeric

Type escape sequence to abort.
Tracing the route to 4.0.0.1

  1 13.0.0.1 20 msec 36 msec 8 msec
  2 10.0.0.2 12 msec 16 msec 20 msec
  3 11.0.0.4 20 msec *  28 msec
```

Por default, RSVP está deshabilitado en todos los routers.

```
2001#show ip rsvp
RSVP: disabled (not enabled on any interface)
Rate Limiting: disabled
  Max msgs per interval: 4
  Interval length (msec): 20
  Max queue size: 500
  Max msgs per second: 200

Refresh Reduction: disabled
  ACK delay (msec): 250
  Initial retransmit delay (msec): 1000
  Local epoch: 0xB67105
  Message IDs: in use 0, total allocated 0, total freed 0

Neighbors: 0
  RSVP encap: 0 UDP encap: 0 RSVP and UDP encap: 0

Local policy:
COPS:

Generic policy settings:
    Default policy: Accept all
    Preemption:     Disabled
```

Una vez que el protocolo de ruteo ha convergido se habilita RSVP en las interfaces de los routers considerados parte del dominio IntServ.

```
ROUTER COMMAND (ip rsvp bandwidth)

router(config-if)#ip rsvp bandwidth [interface-kbps] [single-flow-kbps]
```

RSVP debe ser habilitado en las interfaces de forma específica. El primer parámetro `interface-kbps` es el ancho de banda digital reservado para el protocolo. El segundo es el máximo que un flow puede reservar: `single-flow-kbps`.

```
2001(config)#
!
interface Serial2/1
 ip rsvp bandwidth 50 10
!
interface Serial2/0
 ip rsvp bandwidth 50 10
!
end
```

```
2002(config)#
!
interface Serial2/0
 ip rsvp bandwidth 50 10
!
interface Serial2/1
 ip rsvp bandwidth 50 10
!
end
```

```
2003(config)#
!
interface Serial3/0
 ip rsvp bandwidth 50 10
!
end
```

```
2004(config)#
!
interface Serial2/1
 ip rsvp bandwidth 50 10
!
end
```

RSVP utiliza los algoritmos de encolado y descarte en las interfaces de salida. En el ejemplo se utiliza de forma implícita WFQ. El comando de configuración es el siguiente:

```
ROUTER COMMAND (fair-queue)

router(config-if)#fair-queue [congestive-discard-threshold
                              [dynamic-queues
                              [reservable-queues] ] ]
```

El primer parámetro del comando indica el umbral de descarte, conocido como CDT Congestive Discard Threshold, configurando el número de paquetes máximo en todas las colas de la estructura WFQ que se permiten antes de comenzar a descartar. El segundo indica la cantidad máxima de colas dinámicas de acuerdo a los flujos que manejará la interfaz del router, y el tercero, la cantidad de colas reservables para protocolos de control, como este caso, RSVP. El router tratará de balancear por flujos considerando las prioridades mediante el campo de DSCP, de allí WFQ, aunque los paquetes considerados por los protocolos de control o señalización, los encolará en los buffers reservados.

```
2001#show running-config interface Serial 2/0
!
interface Serial2/0
 description ####_2002.S2/0_####
 ip address 10.0.0.1 255.255.255.0
 fair-queue 64 256
 serial restart-delay 0
 ip rsvp bandwidth 50 10
end
```

Ahora se puede verificar que RSVP está activo en las interfaces.

```
2001#show ip rsvp
RSVP: enabled (on 2 interface(s))
Rate Limiting: disabled
  Max msgs per interval: 4
  Interval length (msec): 20
  Max queue size: 500
  Max msgs per second: 200

Refresh Reduction: disabled
  ACK delay (msec): 250
  Initial retransmit delay (msec): 1000
  Local epoch: 0xB67105
  Message IDs: in use 0, total allocated 0, total freed 0

Neighbors: 0
  RSVP encap: 0 UDP encap: 0 RSVP and UDP encap: 0

Local policy:
COPS:

Generic policy settings:
    Default policy: Accept all
    Preemption:     Disabled

2001#show ip rsvp interface
interface    allocated  i/f max  flow max sub max
Se2/1        0          50K      10K      0
Se2/0        0          50K      10K      0

2002#show ip rsvp interface
interface    allocated  i/f max  flow max sub max
Se2/0        0          50K      10K      0
Se2/1        0          50K      10K      0

2003#show ip rsvp interface
interface    allocated  i/f max  flow max sub max
Se3/0        0          50K      10K      0

2004#show ip rsvp interface
interface    allocated  i/f max  flow max sub max
Se2/1        0          50K      10K      0
```

Los comandos de monitoreo pueden ser los siguientes:

```
ROUTER COMMAND (show ip rsvp ...)

router#show ip rsvp interface [type number]
router#show ip rsvp installed [type number]
router#show ip rsvp neighbor [type number]
router#show ip rsvp sender [type number]
router#show ip rsvp request [type number]
router#show ip rsvp reservation [type number]
```

El comando mostrará la información de RSVP en las interfaces donde está habilitado. Una vez activo se pueden hacer las pruebas. Primero se verá que no hay flujos activos que hayan realizado reserva de recursos.

```
2001#show ip rsvp sender
To              From Pro DPort Sport Prev Hop        I/F      BPS

2002#show ip rsvp reservation
To            From   Pro DPort Sport Next Hop      I/F      Fi Serv BPS
```

```
2002#show ip rsvp request
To            From    Pro DPort Sport Next Hop      I/F      Fi Serv BPS

2001#show ip rsvp neighbor
```

Para las pruebas, y debido a la falta de una aplicación con RSVP, se utilizará uno de los routers como origen del tráfico RSVP y de las datos. Con el siguiente comando se simula la generación del mensaje **RSVP Path**.

```
ROUTER COMMAND (ip rsvp sender-host)

router(config-if)#ip rsvp sender-host session-ip-address sender-ip-address
                                      protocol session-dport sender-sport
                                      bandwidth burst-size
```

- `session-ip-address` dirección IP del receptor, puede ser, unicast o multicast.
- `sender-ip-address` dirección IP del emisor, debe ser una localmente configurada.
- `protocol` puede ser, `{tcp|udp|ip-protocol}`, o un valor entre 0 y 255.
- `session-dport, sender-sport` puertos utilizados para TCP o UDP. Para aplicaciones que no lo usan debe indicarse 0 (cero).
- `bandwidth` ancho de banda promedio reservado en kbps.
- `burst-size` tamaño máximo de ráfaga expresado en KB de datos en cola.

Al probar el comando y chequear el estado se obtiene la siguiente información.

```
2001(config)#ip rsvp sender-host 4.0.0.1 1.0.0.1 TCP 23 0 5 20

2001#show ip rsvp sender
To              From     Pro DPort Sport Prev Hop   I/F      BPS
4.0.0.1         1.0.0.1  TCP 23    0     1.0.0.1              5K
2001#show ip rsvp reservation
To            From      Pro DPort Sport Next Hop   I/F      Fi Serv BPS

2001#show ip rsvp request
To            From      Pro DPort Sport Next Hop   I/F      Fi Serv BPS
```

En los routers se ve el estado de establecimiento de camino hasta el destino.

```
2002#show ip rsvp sender
To              From     Pro DPort Sport Prev Hop     I/F    BPS
4.0.0.1         1.0.0.1  TCP 23    0     10.0.0.1     Se2/0  5K
2002#show ip rsvp reservation
To            From      Pro DPort Sport Next Hop     I/F    Fi Serv BPS
2002#show ip rsvp request
To            From           Pro DPort Sport Next Hop I/F    Fi Serv BPS

2004#show ip rsvp sender
To              From     Pro DPort Sport Prev Hop    I/F      BPS
4.0.0.1         1.0.0.1  TCP 23    0     11.0.0.2    Se2/1    5K
2004#show ip rsvp reservation
To            From      Pro DPort Sport Next Hop    I/F      Fi Serv BPS
2004#show ip rsvp request
To            From      Pro DPort Sport Next Hop    I/F      Fi Serv BPS
```

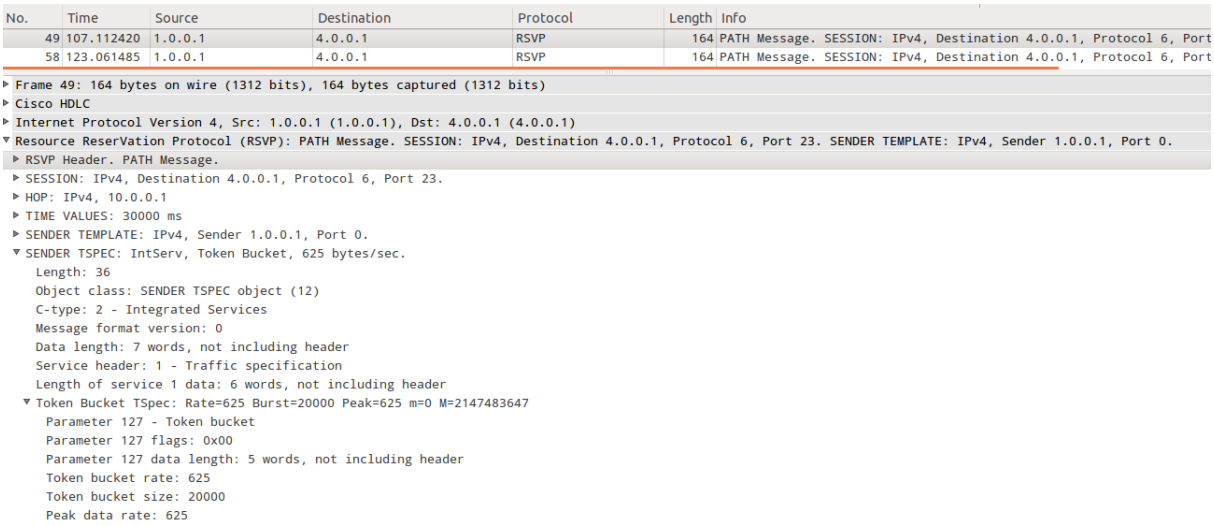


Figura B.2: Estructura del RSVP Path message, vista 1

En las figuras B.2 y B.3 se muestra la estructura del mensaje RSVP Path, se ve que es encapsulado directamente en IP, que va de extremo a extremo, en el ejemplo de 1.0.0.1 a 4.0.0.1. En la sesión se solicita TCP al port 23 al destino 4.0.0.1, luego contiene los campos, Sender Template, Sender Tspec y Adspec. En el campo Sender Tspec se ve el token bucket requerido: 5Kbps, 0.625KBps o 625Bps y ráfaga de 20Kbps o 20000bps. y en Adspec se observa los parámetros que va recolectando, a lo largo del camino, por ejemplo, los nodos que soportan IS, el MTU, además del servicio solicitado, en el ejemplo, de carga controlada.

Luego se ven las direcciones IP de las interfaces desde las cual se hicieron las envíos de los comandos del protocolo de control RSVP.

```
2001#show ip rsvp neighbor
 0.0.0.0         Unknown
 10.0.0.2        Unknown

2002#show ip rsvp neighbor
 0.0.0.0         Unknown
 10.0.0.1        RSVP
 11.0.0.4        Unknown

2004#show ip rsvp neighbor
 4.0.0.1         Unknown
 11.0.0.2        RSVP
```

Previo a esto ejecutando en el router intermedio el debug se pueden ver los logs de los mensajes que pasaron o se generaron.

```
2002#debug ip rsvp
*Apr  4 10:53:09.927: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Received Path message from 10.0.0.1 (on Serial2/0)
*Apr  4 10:53:16.543: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Refresh Path psb = 65AF1F54 refresh interval = 30000mSec
*Apr  4 10:53:16.547: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Sending Path message to 11.0.0.4
```

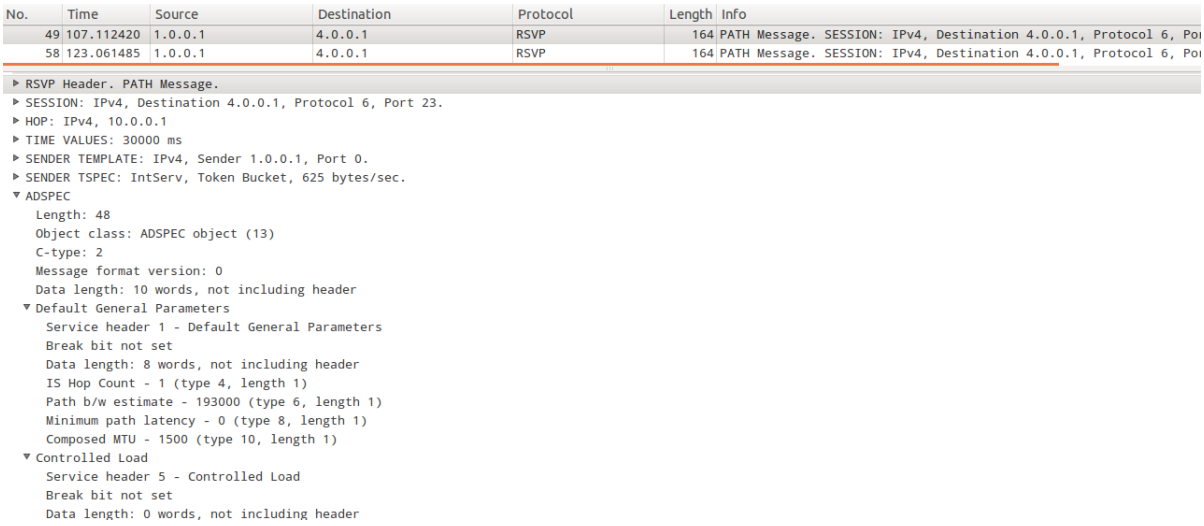


Figura B.3: Estructura del RSVP Path message, vista 2

Si se "niega" el comando inicial se puede ver el mensaje de **Path Teardown**.

```
2001(config)#no ip rsvp sender-host 4.0.0.1 1.0.0.1 TCP 23 0 5 20

2002#
*Apr  4 10:58:00.919: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      PATH TEAR message for 4.0.0.1 (Serial2/0) from 1.0.0.1
*Apr  4 10:58:00.923: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]: remove
      Serial2/0 PATH 4.0.0.1(23) <- 1.0.0.1(6:0)
*Apr  4 10:58:00.927: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Sending PATH TEAR message to 11.0.0.4
```

Se muestra en la figura B.4 el mensaje que da de baja el camino.
Ahora no hay estado generado en el router.

```
2001#show ip rsvp sender
To              From     Pro DPort Sport Prev Hop        I/F      BPS
```

Ejecutándolo nuevamente el comando a continuación.

```
2001(config)#ip rsvp sender-host 4.0.0.1 1.0.0.1 TCP 23 0 5 20
```

Se vuelve a hacer el envío del mensaje para establecer el camino.

```
2002#show ip rsvp sender detail
PATH Session address: 4.0.0.1, port: 23. Protocol: TCP
  Sender address: 1.0.0.1, port: 0
     Inbound from: 10.0.0.1 on interface: Serial2/0
    Traffic params - Rate: 5K bits/sec, Max. burst: 20K bytes
                     Min Policed Unit: 0 bytes,
                     Max Pkt Size 2147483647 bytes
  Path ID handle: 35000402.
  Incoming policy: Accepted. Policy source(s): Default
  Status:
  Output on Serial2/1. Policy status: Forwarding. Handle: 03000400
    Policy source(s): Default

2004#show ip rsvp sender detail
PATH Session address: 4.0.0.1, port: 23. Protocol: TCP
```

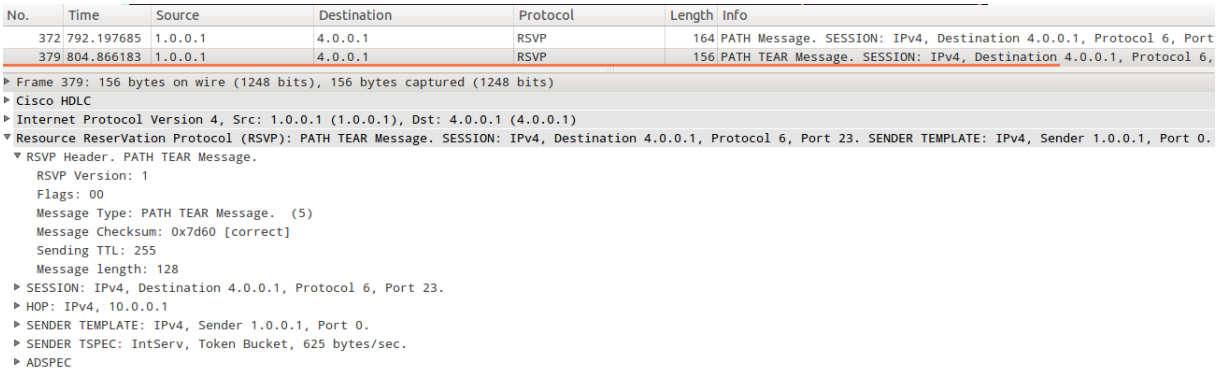


Figura B.4: Estructura del RSVP Teardown message

```
  Sender address: 1.0.0.1, port: 0
     Inbound from: 11.0.0.2 on interface: Serial2/1
  Traffic params - Rate: 5K bits/sec, Max. burst: 20K bytes
                   Min Policed Unit: 0 bytes,
                   Max Pkt Size 2147483647 bytes
  Path ID handle: 1B000402.
  Incoming policy: Accepted. Policy source(s): Default
  Status:
  Output on Loopback0. Policy status: NOT Forwarding. Handle: 03000400
    Policy source(s):
```

Los valores usados son los vistos anteriormente, un data rate de 5Kbps o 0.625KBps y ráfagas de 20Kbps o 20000bps. Hasta ahora solo se estableció, se dio de baja y se volvió a establecer el camino que seguirán los paquetes del flujo. Para hacer la reserva se deben ejecutar los comandos desde el extremo receptor para reservar puntualmente los recursos.

```
ROUTER COMMAND (ip rsvp reservation-host)

router(config)#ip rsvp reservation-host session-ip-address
                        sender-ip-address protocol
                        session-dport sender-sport
                        resv-style service bandwidth burst-size
```

Donde los parámetros del comando son:

- `session-ip-address` dirección IP del receptor, puede ser unicast o multicast. Requiere estar en el router configurada.
- `sender-ip-address` dirección del origen de los datos del flujo. Desde donde se recibirán los datos.
- `protocol` puede ser, `{tcp|udp|ip-protocol}`, igual que para el comando anterior para establecer el camino.

- `session-dport, sender-sport`, igual que para el comando anterior para establecer el camino.

  `resv-style` puede ser, `ff|se|wf`. `ff` significa Fixed Filter, una sola reserva, `se` significa Shared Explicit, reserva compartida limitada al alcance y `wf` significa Wildcard Filter, reserva compartida ilimitada con respecto al "matching" de flujos.

  `service` pude tener los valores, `rate|load`. Mediante el parámetro se indica cual de los dos modelos se utiliza, QoS guaranteed bit rate service o Controlled load service.

- `bandwidth` ancho de banda promedio reservado en kbps.

- `burst-size` tamaño máximo de ráfaga expresado en KB de datos en cola.

Por ejemplo, se podría invocar de la siguiente forma donde se intentan reservar 15Kbps.

```
2004(config)# ip rsvp reservation-host 4.0.0.1 1.0.0.1
              tcp 23 0 ff rate 15 5

2004#show ip rsvp request
To            From     Pro DPort Sport Next Hop   I/F      Fi Serv BPS
4.0.0.1       1.0.0.1 TCP 23    0     11.0.0.2   Se2/1    FF RATE 15K

2004#show ip rsvp request detail

RSVP Reservation. Destination is 4.0.0.1, Source is 1.0.0.1,
  Protocol is TCP, Destination port is 23, Source port is 0
  Next Hop is 11.0.0.2, Interface is Serial2/1
  Reservation Style is Fixed-Filter, QoS Service is Guaranteed-Rate
  Average Bitrate is 15K bits/sec, Maximum Burst is 5K bytes
  Request ID handle: 0A000403.
  Policy: Forwarding. Policy source(s): Default
  PSB Handle List [1 elements]: [0x1B000402]
  RSB Handle List [1 elements]: [0x2000405]
```

En la salida de consola del router con el comando `debug` se pueden ver los logs de los mensajes **RSVP RESV**, aunque no pueden ser servidos, el control de admisión lo rechaza debido a los 15Kbps o 1875Bps que solicita, teniendo configurado un máximo de 10Kbps o 1250Bps por flow. El mensaje de **REJECT** lo hace con un paquete **RSVP ERROR**. La estructura del mensaje se puede ver en la imagen B.5 tomada en el vínculo entre los routers identificados como 2002 y 2004. Se ve también el mensaje de Path, el de Reserva, los refresh de Path y el de Tear de reserva. En los routers no aparece la reserva.

```
*Apr  4 11:04:49.535: RSVP session 4.0.0.1_23[0.0.0.0]:
      Received RESV for 4.0.0.1 (Serial2/1) from 11.0.0.4
*Apr  4 11:04:49.539: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      this RESV has a confirm object
*Apr  4 11:04:49.543: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
```

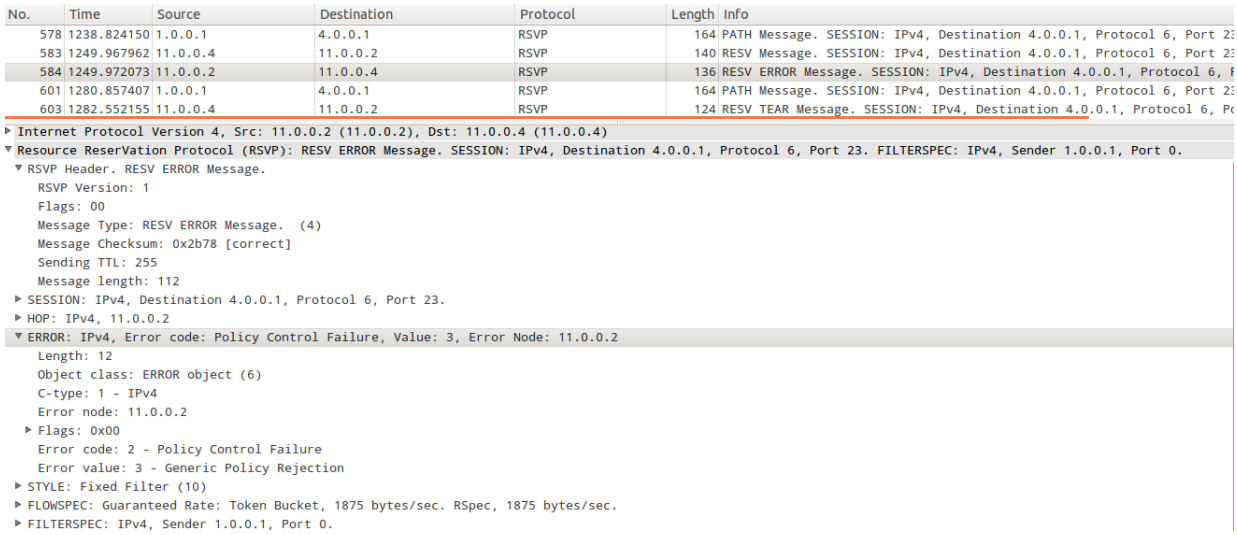


Figura B.5: Estructura del RSVP Error message

```
      reservation not found--new one
*Apr  4 11:04:49.547: RSVP-RESV: Admitting new reservation:65B10BA4
*Apr  4 11:04:49.559: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Sending RESV ERROR message to 11.0.0.4
*Apr  4 11:04:49.563: RSVP-RESV: reservation was not installed:65B10BA4
*Apr  4 11:04:49.567: RSVP-RESV: Deleting reservation: 65B10BA4
*Apr  4 11:04:49.571: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      remove Serial2/1 RESV 4.0.0.1(23) <- 1.0.0.1(6:0)

2002#show ip rsvp request
To            From Pro DPort Sport Next Hop  I/F      Fi Serv BPS

2002#show ip rsvp reservation
To            From Pro DPort Sport Next Hop  I/F      Fi Serv BPS
```

Tirando abajo el requerimiento que no se puede servir se ve en los logs la baja.

```
2004(config)#no ip rsvp reservation-host 4.0.0.1 1.0.0.1
                      tcp 23 0 ff rate 15 5

2002#
*Apr  4 11:05:58.859: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      RESV TEAR message for 4.0.0.1 (Serial2/1) from 11.0.0.4
*Apr  4 11:05:58.859: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      RESV TEAR message  for 4.0.0.1 (Serial2/1) from 11.0.0.4

2004#show ip rsvp request detail
```

Ahora enviando el requerimiento en concordancia con los 10Kbps o 1250Bps configurados se tiene éxito en la reserva lo cual se puede apreciar en los logs. Se solicita en el mismo paquete un máximo de ráfaga de 20Kbps o 2500Bps.

```
2004(config)#ip rsvp reservation-host 4.0.0.1 1.0.0.1
                    tcp 23 0 ff rate 10 20

*Apr  4 11:11:05.951: RSVP session 4.0.0.1_23[0.0.0.0]:
      Received RESV for 4.0.0.1 (Serial2/1) from 11.0.0.4
*Apr  4 11:11:05.959: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      this RESV has a confirm object
*Apr  4 11:11:05.959: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
```

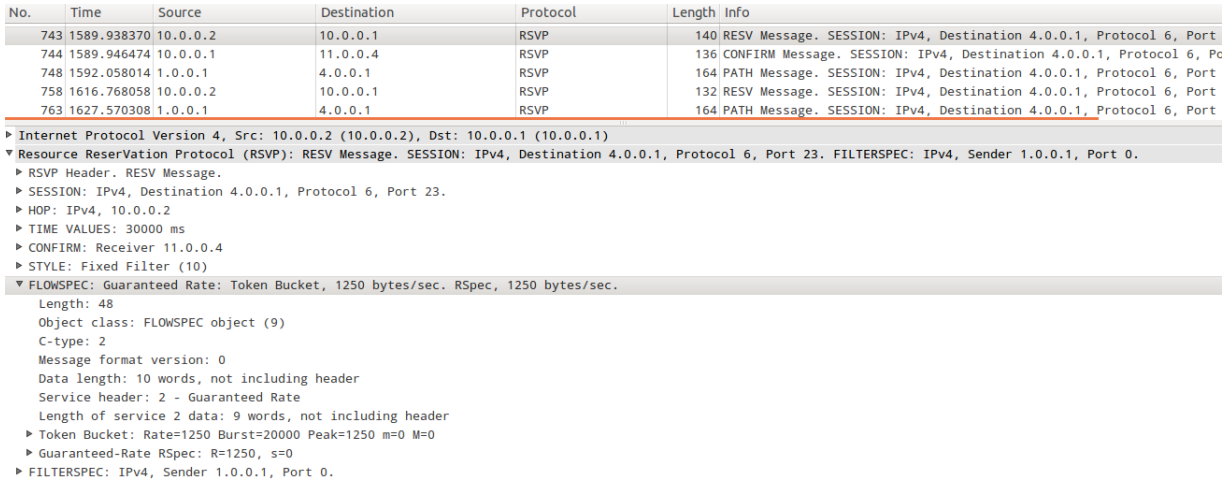


Figura B.6: Estructura del RSVP Resv message y la confirmación

```
      reservation not found--new one
*Apr  4 11:11:05.967: RSVP-RESV: Admitting new reservation:65B10BA4
*Apr  4 11:11:05.975: RSVP-RESV: reservation was installed:65B10BA4
*Apr  4 11:11:05.979: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      start requesting 10 kbps FF  reservation for 1.0.0.1(0)
      TCP-> 4.0.0.1(23) on Serial2/0 neighbor 10.0.0.1
*Apr  4 11:11:05.983: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Refresh RESV, req=672C0CF0, refresh interval=0mSec
      [cleanup timer is not awake]
*Apr  4 11:11:05.983: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Sending Resv message to 10.0.0.1
*Apr  4 11:11:05.991: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      RESV CONFIRM Message for 4.0.0.1 (Serial2/0) from 10.0.0.1
*Apr  4 11:11:05.991: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Sending RESV CONFIRM message to 11.0.0.4
*Apr  4 11:11:08.107: RSVP 1.0.0.1_0->4.0.0.1_23[0.0.0.0]:
      Received Path message from 10.0.0.1 (on Serial2/0)
```

En la figura B.6 se muestra el mensaje de reserva **RSVP RESV message**, con la confirmación. El mensaje de reserva contiene los campos: Estilo de reserva, aquí, fijo; el campo Flowspec donde solicita 10Kbps con un servicio "2" sin slack term y el Filterspec indicando el origen.

Al ejecutar los comandos en los routers se puede ver el estado. En el destino se detectan el paso de los mensajes por el camino establecido anteriormente, con una capacidad del emisor indicada en 5Kbps. Se ven también el paso de los mensajes de reserva realizada a lo largo del trayecto con un rate de 10Kbps.

```
2004#show ip rsvp sender
To              From     Pro DPort Sport Prev Hop  I/F      BPS
4.0.0.1         1.0.0.1  TCP 23    0     11.0.0.2  Se2/1    5K

2004#show ip rsvp request
To              From     Pro DPort Sport Next Hop  I/F      Fi Serv BPS
4.0.0.1         1.0.0.1  TCP 23    0     11.0.0.2  Se2/1    FF RATE 10K

2004#show ip rsvp reservation
To              From     Pro DPort Sport Next Hop  I/F      Fi Serv BPS
4.0.0.1         1.0.0.1  TCP 23    0     4.0.0.1             FF RATE 10K
```

```
2004#show ip rsvp installed
RSVP: Serial2/1 has no installed reservations
```

En los routers a lo largo del camino, ubicados a continuación del receptor se observa la reserva instalada ejecutando los siguientes comandos.

```
2002#show ip rsvp sender
To              From    Pro DPort Sport Prev Hop  I/F      BPS
4.0.0.1         1.0.0.1 TCP 23    0     10.0.0.1  Se2/0    5K

2002#show ip rsvp request
To              From      Pro DPort Sport Next Hop  I/F      Fi Serv BPS
4.0.0.1         1.0.0.1   TCP 23    0     10.0.0.1  Se2/0    FF RATE 10K

2002#show ip rsvp reservation
To              From      Pro DPort Sport Next Hop  I/F      Fi Serv BPS
4.0.0.1         1.0.0.1   TCP 23    0     11.0.0.4  Se2/1    FF RATE 10K

2002#sh ip rsvp installed
RSVP: Serial2/0 has no installed reservations
RSVP: Serial2/1
BPS    To              From    Protoc DPort  Sport  Weight Conversation
10K    4.0.0.1         1.0.0.1 TCP    23     0      6      265

2002#show ip rsvp installed detail
RSVP: Serial2/0 has no installed reservations

RSVP: Serial2/1 has the following installed reservations
RSVP Reservation. Destination is 4.0.0.1. Source is 1.0.0.1,
  Protocol is TCP, Destination port is 23, Source port is 0
  Traffic Control ID handle: 11000402
  Created: 17:25:01 UTC Sat Apr 9 2011
  Compression: (rtp compression not predicted:
  IPHC internal error: invalid input)
  Admitted flowspec:
    Reserved bandwidth: 10K bits/sec, Maximum burst: 20K bytes,
    Peak rate: 10K bits/sec Min Policed Unit: 0 bytes,
    Max Pkt Size: 0 bytes
  Resource provider for this flow:
    WFQ on hw idb Se2/1:  RESERVED queue 265.  Weight: 6, BW 10 kbps
  Conversation supports 1 reservations [0x4000404]
  Data given reserved service: 0 packets (0 bytes)
  Data given best-effort service: 0 packets (0 bytes)
  Reserved traffic classified for 225 seconds
  Long-term average bitrate (bits/sec): 0 reserved, 0 best-effort
  Policy: INSTALL. Policy source(s): Default

2001#show ip rsvp sender
To              From    Pro DPort Sport Prev Hop  I/F      BPS
4.0.0.1         1.0.0.1 TCP 23    0     1.0.0.1             5K

2001#show ip rsvp request
To              From      Pro DPort Sport Next Hop  I/F      Fi Serv BPS

2001#show ip rsvp reservation
To              From      Pro DPort Sport Next Hop  I/F      Fi Serv BPS
4.0.0.1         1.0.0.1   TCP 23    0     10.0.0.2  Se2/0    FF RATE 10K

2001#show ip rsvp installed
RSVP: Serial2/1 has no installed reservations
RSVP: Serial2/0
BPS    To              From    Protoc DPort  Sport  Weight Conversation
10K    4.0.0.1         1.0.0.1 TCP    23     0      6      265
```

Si se inspecciona el estado de las colas en las interfaces por donde pasará el flujo de los datos reservados vía RSVP se muestra como tráfico no "captado" por WFQ.

En este caso RSVP se ve en las colas reservadas. De igual manera que se vería el tráfico reservado, como sería el de control y el de protocolos de señalización.

```
2001#show queue Serial 2/0
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 1/1 (allocated/max allocated)
     Available Bandwidth 1148 kilobits/sec
```

Las pruebas se realizan sobre IPv4 debido a que la plataforma con IntServ/RSVP no soporta IPv6.

Apéndice C

Ejemplos con disciplinas de encolado

En este apéndice se mostrarán algunos casos de uso y configuraciones a modo de ejemplo con colas de paquetes sobre routers. Se mostrarán sobre la plataforma de router cisco y sobre el sistema operativo GNU/Linux. Las pruebas serán con flujos generados de acuerdo a la figura C.1.

C.1. Ejemplo de WFQ

El primer ejemplo es sobre un política WFQ sobre routers cisco, donde las colas de acuerdo, a los flujos detectados se formarán de manera automática. El ejemplo se muestra sobre IPv6, aunque también fue realizado con IPv4. Primero se configura el ruteo en la red y se prueba que el origen tenga conectividad. No se muestran todos los detalles de los comandos, pues sino el texto sería demasiado extenso. Primero se presenta la configuración de OSPF en un nodo de la red y como se ve la tabla RIB en

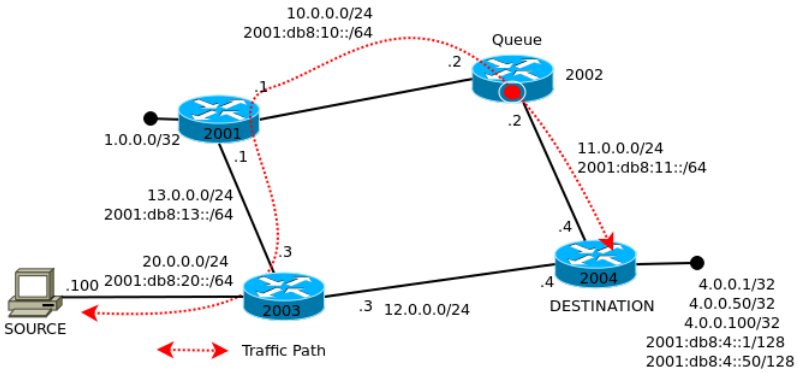


Figura C.1: Topología de prueba para ver ejemplos de políticas de encolado

otro equipo. Para IPv6 la configuración de las redes no se realiza dentro del comando `network`, sino de forma más natural, en las interfaces.

```
2002#

ipv6 unicast-routing

interface Serial2/0
 description ####_2001.Serial2/0_####
 ip address 10.0.0.2 255.255.255.0
 ipv6 address 2001:DB8:10::2/64
 ipv6 ospf 1 area 0.0.0.0
 serial restart-delay 0
!
interface Serial2/1
 description ####_2004.Serial2/1_####
 ip address 11.0.0.2 255.255.255.0
 ipv6 address 2001:DB8:11::2/64
 ipv6 ospf 1 area 0.0.0.0
 serial restart-delay 0
!

ipv6 router ospf 1
```

La configuración de OSPFv3 ya proporcionó la convergencia.

```
2001#show ipv6 ospf neighbor

Neighbor ID     Pri   State       Dead Time   Interface ID   Interface
2.0.0.1           1   FULL/  -    00:00:38    5              Serial2/0
3.0.0.1           1   FULL/  -    00:00:38    15             Serial2/1

2001#show ipv6 route
IPv6 Routing Table - 11 entries
Codes: C - Connected, L - Local, S - Static, R - RIP, B - BGP
       U - Per-user Static route, M - MIPv6
       I1 - ISIS L1, I2 - ISIS L2, IA - ISIS interarea,
       IS - ISIS summary O - OSPF intra, OI - OSPF inter,
       OE1 - OSPF ext 1, OE2 - OSPF ext 2
       ON1 - OSPF NSSA ext 1, ON2 - OSPF NSSA ext 2
       D - EIGRP, EX - EIGRP external
O   2001:DB8:4::1/128 [110/128]
     via FE80::C805:10FF:FEFF:0, Serial2/0
O   2001:DB8:4::50/128 [110/128]
     via FE80::C805:10FF:FEFF:0, Serial2/0
O   2001:DB8:4::100/128 [110/128]
     via FE80::C805:10FF:FEFF:0, Serial2/0
C   2001:DB8:10::/64 [0/0]
     via ::, Serial2/0
L   2001:DB8:10::1/128 [0/0]
     via ::, Serial2/0
O   2001:DB8:11::/64 [110/128]
     via FE80::C805:10FF:FEFF:0, Serial2/0
O   2001:DB8:12::/64 [110/65]
     via FE80::C806:10FF:FEFF:0, Serial2/1
C   2001:DB8:13::/64 [0/0]
     via ::, Serial2/1
L   2001:DB8:13::1/128 [0/0]
     via ::, Serial2/1
O   2001:DB8:20::/64 [110/74]
     via FE80::C806:10FF:FEFF:0, Serial2/1
L   FF00::/8 [0/0]
     via ::, Null0
```

A continuación se configura desde donde se iniciarán los flujos y se ve que se llega al destino. Se marcan las diferentes direcciones IP que tendrá el origen, 3 globales y una de link-local.

```
root@source:~# ip -f inet6 addr add 2001:DB8:20::100/64 dev tap0

root@source:~# ip -f inet6 addr show dev tap0
7: tap0: <BROADCAST,MULTICAST> mtu 1500 qlen 500
    inet6 2001:db8:20::100/64 scope global tentative
       valid_lft forever preferred_lft forever

root@source:~# ip -f inet6 addr show dev tap0
7: tap0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qlen 500
    inet6 2001:db8:20:0:716d:4997:9d50:f7c0/64
    scope global temporary dynamic
       valid_lft 604795sec preferred_lft 85795sec
    inet6 2001:db8:20:0:7cc1:a9ff:feb8:efdc/64
    scope global dynamic
       valid_lft 2591995sec preferred_lft 604795sec
    inet6 2001:db8:20::100/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::7cc1:a9ff:feb8:efdc/64 scope link
       valid_lft forever preferred_lft forever

root@source:~# ip -f inet6 route show
2001:db8:20::/64 dev tap0  proto kernel  metric 256
fe80::/64 dev wlan0  proto kernel  metric 256
fe80::/64 dev tap0  proto kernel  metric 256
default via fe80::2022:22ff:fe22:2222 dev tap0  proto kernel
            metric 1024  expires 1628sec
```

Directamente probando la red.

```
root@source:~# traceroute6 -n 2001:DB8:4::1
traceroute to 2001:DB8:4::1 (2001:db8:4::1)
 from 2001:db8:20:0:716d:4997:9d50:f7c0, 30 hops max, 24byte packets
 1  2001:db8:20::3  1.733 ms  4.159 ms  2.123 ms
 2  2001:db8:13::1  8.418 ms  8.499 ms  11.34 ms
 3  2001:db8:10::2  12.651 ms  12.601 ms  17.61 ms
 4  2001:db8:4::1  20.929 ms  16.71 ms  20.863 ms

root@source:~# traceroute6 -n -s 2001:db8:20::100 2001:DB8:4::1
traceroute to 2001:DB8:4::1 (2001:db8:4::1)
 from 2001:db8:20::100, 30 hops max, 24 byte packets
 1  2001:db8:20::3  1.95 ms  2.182 ms  4.403 ms
 2  2001:db8:13::1  8.284 ms  8.604 ms  10.611 ms
 3  2001:db8:10::2  12.727 ms  15.719 ms  12.781 ms
 4  2001:db8:4::1  20.067 ms  18.774 ms  16.862 ms

root@source:~# traceroute6 -n -s 2001:db8:20::100 2001:DB8:4::50
traceroute to 2001:DB8:4::50 (2001:db8:4::50)
 from 2001:db8:20::100, 30 hops max, 24 byte packets
 1  2001:db8:20::3  3.924 ms  4.173 ms  4.306 ms
 2  2001:db8:13::1  6.314 ms  8.381 ms  10.477 ms
 3  2001:db8:10::2  13.045 ms  15.386 ms  12.866 ms
 4  2001:db8:4::50  23.166 ms  21.031 ms  18.936 ms

root@source:~# traceroute6 -n -s 2001:db8:20::100 2001:DB8:4::100
traceroute to 2001:DB8:4::100 (2001:db8:4::100)
 from 2001:db8:20::100, 30 hops max, 24 byte packets
 1  2001:db8:20::3  58.654 ms  2.587 ms  2.258 ms
 2  2001:db8:13::1  12.857 ms  7.044 ms  8.399 ms
 3  2001:db8:10::2  15.902 ms  14.95 ms  14.669 ms
 4  2001:db8:4::100  18.764 ms  18.974 ms  16.739 ms
```

Primero se puede ver que al sacar la política default la interfaz se configura sobre una política de encolado FIFO. En general, para los routers cisco, de fábrica, las interfaces de baja velocidad tienen una política WFQ y para velocidades de datos iguales o mayores a 10Mbps una FIFO. Por ejemplo, para interfaces seriales vamos a encontrar

WFQ si no se configura nada extra. En este caso se inspecciona la configuración sobre donde entrará el tráfico desde el origen, la interfaz `Serial2/0`. Luego será cambiada por la disciplina WFQ. Podría ser más adecuado configurarla en `Serial2/1` pero en el ejemplo, como el tráfico es ida y vuelta, funcionará de igual manera.

```
2002(config)#interface S2/0
2002(config-if)#no fair-queue

2002#show int S2/0
  ...
    Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/40 (size/max)
  ...

2002#show queue S2/0
'Show queue' not supported with FIFO queueing.
```

Pasar a un modelo WFQ. Al hacerlo se ve la cantidad máxima de colas asociadas a los flows en el valor 256, por las cuales se ha detectado en algún momento que se utilizado solo una.

```
2002(config)#interface S2/0
2002(config-if)#fair-queue

2002#show int S2/0
  ...
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     ...

2002#show queue S2/0
  Input queue: 0/75/0/0 (size/max/drops/flushes);Total output drops:0
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/0 (size/max total/threshold/drops)
     Conversations  0/1/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 1158 kilobits/sec
```

Se generan las conexiones y tráfico con `telnet(1)` y ejecutando `ping6(8)` para que los diferentes paquetes sean asociados a los flujos específicos. Seguidamente, inspeccionar como se detectan los mismos.

```
root@source:~# telnet 2001:DB8:4::1
...
root@source:~# ping6 -A -s 1000 2001:DB8:4::50
...
root@source:~# ping6 -A -s 1000 2001:DB8:4::1
...
root@source:~# telnet 2001:DB8:4::100
...
```

Para luego inspeccionar la cola y los flujos encontrados.

```
2002#show queue S2/0
 Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
 Queueing strategy: weighted fair
 Output queue: 5/1000/64/0 (size/max total/threshold/drops)
    Conversations  3/3/256 (active/max active/max total)
    Reserved Conversations 0/0 (allocated/max allocated)
    Available Bandwidth 1158 kilobits/sec
```

```
 (depth/weight/total drops/no-buffer drops/interleaves) 1/4626/0/0/0
 Conversation 28, linktype: ipv6, length: 65
 source: 2001:DB8:4::1, destination: 2001:DB8:20:0:716D:4997:9D50:F7C0
 hop-limit: 63, TC: 192, prot: 6,source port 23,destination port 34852

 (depth/weight/total drops/no-buffer drops/interleaves) 3/32384/0/0/0
 Conversation 141, linktype: ipv6, length: 108
 source: 2001:DB8:4::1, destination: 2001:DB8:20:0:716D:4997:9D50:F7C0
 hop-limit: 63, TC: 0, prot: 58

 (depth/weight/total drops/no-buffer drops/interleaves) 1/32384/0/0/0
 Conversation 220, linktype: ipv6, length: 108
 source: 2001:DB8:4::50,destination: 2001:DB8:20:0:716D:4997:9D50:F7C0
 hop-limit: 63, TC: 0, prot: 58
 ...

2002#show queue S2/0
 Input queue: 0/75/0/0 (size/max/drops/flushes);
 Total output drops: 1237
 Queueing strategy: weighted fair
 Output queue: 65/1000/64/1237 (size/max total/threshold/drops)
    Conversations  3/4/256 (active/max active/max total)
    Reserved Conversations 0/0 (allocated/max allocated)
    Available Bandwidth 1158 kilobits/sec

 (depth/weight/total drops/no-buffer drops/interleaves) 1/4626/0/0/0
 Conversation 28, linktype: ipv6, length: 65
 source: 2001:DB8:4::1,destination: 2001:DB8:20:0:716D:4997:9D50:F7C0,
 hop-limit: 63,TC: 192,prot: 6,source port 23,destination port 34852

 (depth/weight/total drops/no-buffer drops/interleaves) 1/4626/0/0/0
 Conversation 59, linktype: ipv6, length: 65
 source: 2001:DB8:4::100, destination:2001:DB8:20:0:716D:4997:9D50:F7C0
 hop-limit: 63,TC: 192,prot: 6,source port 23, destination port 54519

 (depth/weight/total drops/no-buffer drops/interleaves)63/32384/1237/0/
 Conversation 141, linktype: ipv6, length: 108
 source: 2001:DB8:4::1, destination: 2001:DB8:20:0:716D:4997:9D50:F7C0,
 hop-limit: 63, TC: 0, prot: 58
```

Finalmente se pueden ver todos los flujos que fueron detectados, aunque no están más activos. En lo mostrado: 4 (cuatro).

```
2002#show queue S2/0
  Input queue: 0/75/0/0 (size/max/drops/flushes);
  Total output drops: 2748
  Queueing strategy: weighted fair
  Output queue: 0/1000/64/2748 (size/max total/threshold/drops)
     Conversations  0/4/256 (active/max active/max total)
     Reserved Conversations 0/0 (allocated/max allocated)
     Available Bandwidth 1158 kilobits/sec
```

El sistema WFQ es muy simple de configurar. De acuerdo a lo visto trata a todo el tráfico por igual. No es recomendado en el núcleo de la red, donde se tienen interfaces de alta velocidad y por el cual pasarán gran cantidad de flujos.

C.2. Ejemplo de PRIO en cisco

El siguiente ejemplo muestra una cola de prioridades estricta, PRIO, en un router cisco. Las colas PRIO solo son aplicables con ACL (Access Control List) en la

plataforma testeada, cisco con tráfico IPv4. Se pueden aplicar en la misma a IPv6, pero la configuración de condiciones de selección de tráfico se limitan solo a poder hacerse por tamaño de paquetes. En el ejemplo se seleccionan los paquetes específicos con ACL sobre IPv4. Este tipo de cola genera una separación y un orden estricto de atención, para cisco son 4 sub-colas, cada una más prioritaria que la siguiente.

```
2002(config)#priority-list 10 protocol ip ?
  high
  medium
  normal
  low

2002(config)#priority-list 10 protocol ipv6 high ?
  gt  Prioritize packets greater than a specified size
  lt  Prioritize packets less than a specified size
  <cr>
```

Al configurarse se crean las cuatro sub-colas y se selecciona el tráfico de acuerdo a las ACL IPv4, o incluso, usando parámetros directamente asignables en la definición.

```
priority-list 10 protocol ip high list 101
priority-list 10 protocol ip medium list 102
priority-list 10 protocol ip normal list 103
priority-list 10 protocol ip low udp 5000
```

Para el ejemplo la configuración se realiza indicando que el tráfico más prioritario es telnet al destino 4.0.0.1, luego ICMP al destino 4.0.0.100 y como tráfico normal o regular el telnet a 4.0.0.5. El tráfico UDP se deja como el de más baja prioridad o default.

```
access-list 101 permit tcp any host 4.0.0.1 eq telnet
access-list 102 permit icmp any host 4.0.0.100
access-list 103 permit tcp any host 4.0.0.50 eq telnet
```

Luego se aplica a la interfaz.

```
interface Serial2/1
 priority-group 10
```

Se puede inspeccionar que esta instalada la cola y ver los parámetros.

```
2002#show interfaces Serial 2/1 | include queue
  Input queue: 0/75/0/0 (size/max/drops/flushes);
  Total output drops: 2748
  Output queue (queue priority: size/max/drops):

2002#show queue Serial 2/1

2002#show queueing priority
Current DLCI priority queue configuration:
Current priority queue configuration:

List   Queue  Args
10     high   protocol ip          list 101
10     medium protocol ip          list 102
10     normal protocol ip          list 103
10     low    protocol ip          udp port 5000
```

Por último, para hacer el testing, se genera el tráfico y se ve como se va encolando el mismo en las diferentes sub-colas de acuerdo a la prioridad definida.

```
2002#clear counters Serial 2/1

2002#show queueing interface S2/1
Interface Serial2/1 queueing strategy: priority

Output queue utilization (queue/count)
high/0 medium/0 normal/0 low/0

root@source:~# telnet 4.0.0.1
...

2002#show queueing interface S2/1
Interface Serial2/1 queueing strategy: priority

Output queue utilization (queue/count)
high/949 medium/0 normal/0 low/0

root@source:/home/andres# telnet 4.0.0.50
...

2002#show queueing interface S2/1
Interface Serial2/1 queueing strategy: priority

Output queue utilization (queue/count)
high/1042 medium/0 normal/140 low/0

root@source:~# ping  4.0.0.100 -s 1000
...

2002#show queueing interface S2/1
Interface Serial2/1 queueing strategy: priority

Output queue utilization (queue/count)
high/1423 medium/18 normal/180 low/0
```

Se ve el "matching" en las ACL asociadas a las sub-colas.

```
2002#show access-lists 101
Extended IP access list 101
    10 permit tcp any host 4.0.0.1 eq telnet (1322 matches)

2002#show access-lists 102
Extended IP access list 102
    10 permit icmp any host 4.0.0.100 (18 matches)

2002#show access-lists 103
Extended IP access list 103
    10 permit tcp any host 4.0.0.50 eq telnet (159 matches)
```

El tráfico UDP va al final, a la sub-cola o clase LOW (BAJA).

```
root@source:~# iperf -u -c 4.0.0.1 -p 5000

002#show queueing interface S2/1
Interface Serial2/1 queueing strategy: priority

Output queue utilization (queue/count)
high/1464 medium/18 normal/186 low/15
```

Si se genera mucho tráfico correspondiente a las colas de las prioridades más alta se puede ver una importante degradación en las que están por debajo, por ejemplo, la segunda sesión de telnet se pone más lenta. Si se pone como menos prioritario el tráfico ICMP se percibirá que se pierden mensajes.
Con el reemplazo de la siguiente configuración.

```
no priority-list 10 protocol ip high list 101
no priority-list 10 protocol ip medium list 102
no priority-list 10 protocol ip normal list 103
no priority-list 10 protocol ip low udp 5000

priority-list 10 protocol ip high list 101
priority-list 10 protocol ip normal list 103
priority-list 10 protocol ip low list 102
```

Se obtiene retardos largos.

```
-- 4.0.0.100 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5007ms
rtt min/avg/max/mdev = 25.825/53.184/175.474/54.723 ms
root@source:~# ping 4.0.0.100 -s 1500
PING 4.0.0.100 (4.0.0.100) 1500(1528) bytes of data.
1508 bytes from 4.0.0.100: icmp_req=1 ttl=252 time=26.4 ms
...
1508 bytes from 4.0.0.100: icmp_req=23 ttl=252 time=98.9 ms
1508 bytes from 4.0.0.100: icmp_req=24 ttl=252 time=291 ms
1508 bytes from 4.0.0.100: icmp_req=25 ttl=252 time=306 ms
1508 bytes from 4.0.0.100: icmp_req=26 ttl=252 time=269 ms
1508 bytes from 4.0.0.100: icmp_req=27 ttl=252 time=297 ms
1508 bytes from 4.0.0.100: icmp_req=28 ttl=252 time=167 ms
1508 bytes from 4.0.0.100: icmp_req=29 ttl=252 time=338 ms
1508 bytes from 4.0.0.100: icmp_req=30 ttl=252 time=284 ms
1508 bytes from 4.0.0.100: icmp_req=31 ttl=252 time=419 ms
1508 bytes from 4.0.0.100: icmp_req=32 ttl=252 time=354 ms
1508 bytes from 4.0.0.100: icmp_req=33 ttl=252 time=346 ms
1508 bytes from 4.0.0.100: icmp_req=34 ttl=252 time=2268 ms
hl1508 bytes from 4.0.0.100: icmp_req=35 ttl=252 time=1434 ms
1508 bytes from 4.0.0.100: icmp_req=36 ttl=252 time=570 ms
1508 bytes from 4.0.0.100: icmp_req=37 ttl=252 time=380 ms
1508 bytes from 4.0.0.100: icmp_req=38 ttl=252 time=2627 ms
1508 bytes from 4.0.0.100: icmp_req=39 ttl=252 time=1776 ms
1508 bytes from 4.0.0.100: icmp_req=40 ttl=252 time=924 ms
1508 bytes from 4.0.0.100: icmp_req=41 ttl=252 time=399 ms
1508 bytes from 4.0.0.100: icmp_req=42 ttl=252 time=332 ms
1508 bytes from 4.0.0.100: icmp_req=43 ttl=252 time=4183 ms
...
--- 4.0.0.100 ping statistics ---
51 packets transmitted, 51 received, 0% packet loss, time 50097ms
rtt min/avg/max/mdev = 20.550/538.513/4183.158/904.291 ms, pipe 5
```

Aunque no alcanzó a producirse descartes.

C.3. Ejemplo de CBWFQ/LLQ en cisco

Para el ejemplo con CBWFQ se muestra primero una configuración para IPv4 y luego para IPv6. Primero se crean las ACL y los `class-map`. Los class-map son los clasificadores asociados a las ACL en el router.

```
2002(config)#
access-list 101 permit tcp any host 4.0.0.1 eq telnet
access-list 102 permit icmp any host 4.0.0.100
access-list 103 permit tcp any host 4.0.0.50 eq telnet

class-map match-all CLASS-UNO
 match access-group 101
class-map match-all CLASS-DOS
 match access-group 102
class-map match-all CLASS-TRES
 match access-group 103
```

Seguidamente se arma la política con los class-map definidos previamente. La política asigna a cada clase, o cola, determinado tratamiento. En este caso se asegura un ancho de banda a cada clase a partir de un porcentaje del total y para la default, la cola CATCH-ALL, se aplica WFQ combinado con RED.

```
policy-map POL-OUT
 class CLASS-UNO
  bandwidth percent 20
 class CLASS-DOS
  bandwidth percent 30
    police 8000
 class CLASS-TRES
  bandwidth percent 10
 class class-default
  fair-queue
  random-detect
```

En el ejemplo, para la clase `CLASS-DOS`, se pone un policy que en la práctica no tiene sentido con el bandwidth otorgado, pero se utilizará para ver como se aplica la política de descarte en las pruebas mostradas. Luego de creada la política se debe aplicar a una interfaz, de igual forma como se vio en los tests anteriores.

```
interface Serial2/0
 service-policy output POL-OUT
 load-interval 30
```

Se limpian los contadores y, por último, queda testearla.

```
2002#clear access-list counters
2002#clear counters S2/1
```

Una vez generados los datos se ve que se encola el tráfico de las diferentes clases en las sub-colas correspondientes.

```
root@source:~# telnet 4.0.0.1
...

root@source:~# ping 4.0.0.100
PING 4.0.0.100 (4.0.0.100) 56(84) bytes of data.
64 bytes from 4.0.0.100: icmp_req=1 ttl=252 time=13.7 ms
64 bytes from 4.0.0.100: icmp_req=2 ttl=252 time=16.5 ms
64 bytes from 4.0.0.100: icmp_req=3 ttl=252 time=14.9 ms
64 bytes from 4.0.0.100: icmp_req=4 ttl=252 time=19.0 ms
64 bytes from 4.0.0.100: icmp_req=5 ttl=252 time=16.3 ms
...

root@source:/home/andres# telnet 4.0.0.50
...

root@source:~# iperf -u -c 4.0.0.1 -p 5000

2002#show policy-map interface S2/1
 Serial2/1

  Service-policy output: POL-OUT

    Class-map: CLASS-UNO (match-all)
      62 packets, 2851 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: access-group 101
```

```
      Queueing
        Output Queue: Conversation 265
        Bandwidth 20 (%)
        Bandwidth 308 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 1/44
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: CLASS-DOS (match-all)
      15 packets, 8220 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: access-group 102
      Queueing
        Output Queue: Conversation 266
        Bandwidth 30 (%)
        Bandwidth 463 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0
      police:
          cir 8000 bps, bc 1500 bytes
        conformed 10 packets, 700 bytes; actions:
          transmit
        exceeded 5 packets, 7520 bytes; actions:
          drop
        conformed 0 bps, exceed 0 bps

    Class-map: CLASS-TRES (match-all)
      101 packets, 4663 bytes
      30 second offered rate 2000 bps, drop rate 0 bps
      Match: access-group 103
      Queueing
        Output Queue: Conversation 267
        Bandwidth 10 (%)
        Bandwidth 154 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 3/132
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: class-default (match-any)
      339 packets, 314236 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
        Flow Based Fair Queueing
        Maximum Number of Hashed Queues 256
        (total queued/total drops/no-buffer drops) 0/0/0
         exponential weight: 9

class    Transmitted  Random drop Tail drop  Minimum Maximum  Mark
         pkts/bytes   pkts/bytes  pkts/bytes  thresh  thresh  prob
    0     260/307616      0/0         0/0         20      40  1/10
    1       0/0           0/0         0/0         22      40  1/10
    2       0/0           0/0         0/0         24      40  1/10
    3       0/0           0/0         0/0         26      40  1/10
    4       0/0           0/0         0/0         28      40  1/10
    5       0/0           0/0         0/0         30      40  1/10
    6      40/3344        0/0         0/0         32      40  1/10
    7      39/3276        0/0         0/0         34      40  1/10
 rsvp       0/0           0/0         0/0         36      40  1/10
```

También se ven los matches en las ACLs.

```
2002#show access-lists 101
Extended IP access list 101
    10 permit tcp any host 4.0.0.1 eq telnet (62 matches)
2002#show access-lists 102
Extended IP access list 102
    10 permit icmp any host 4.0.0.100 (15 matches)
2002#show access-lists 103
Extended IP access list 103
```

```
    10 permit tcp any host 4.0.0.50 eq telnet (103 matches)

002#show policy-map interface S2/1 output
 Serial2/1

  Service-policy output: POL-OUT
    ...

    Class-map: CLASS-DOS (match-all)
      99 packets, 18372 bytes
      30 second offered rate 2000 bps, drop rate 0 bps
      Match: access-group 102
      Queueing
        Output Queue: Conversation 266
        Bandwidth 30 (%)
        Bandwidth 463 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0
      police:
          cir 8000 bps, bc 1500 bytes
        conformed 63 packets, 5292 bytes; actions:
          transmit
        exceeded 36 packets, 13080 bytes; actions:
        drop
        conformed 1000 bps, exceed 0 bps
        ...
```

Se puede cambiar la política de CBWFQ a una LLQ, indicando el parámetro de cual de las colas o clases es la prioritaria para el tráfico sensitivo al delay.

```
policy-map POL-OUT
 class CLASS-UNO
  priority percent 10
 class CLASS-DOS
  bandwidth percent 30
    police 8000
 class CLASS-TRES
  bandwidth percent 10
 class class-default
  fair-queue
  random-detect
```

Previamente se debió des-configurar el parámetro **bandwidth**.

C.4. Ejemplo de CBWFQ sobre IPv6 en cisco

Para ilustrar el uso en IPv6 se muestra la configuración análoga a la anterior, CBWFQ en IPv4, pero para el protocolo IPv6.

```
2002(config)#
ipv6 access-list ACLV6-101
 permit tcp any host  2001:DB8:4::1 eq telnet
ipv6 access-list ACLV6-102
 permit icmp any host  2001:DB8:4::100
ipv6 access-list ACLV6-103
 permit tcp any host 2001:DB8:4::50 eq telnet
 ...

2002#show ipv6 access-list
IPv6 access list ACLV6-101
    permit tcp any host 2001:DB8:4::1 eq telnet sequence 10
IPv6 access list ACLV6-102
```

```
    permit icmp any host 2001:DB8:4::100 sequence 10
IPv6 access list ACLV6-103
    permit tcp any host 2001:DB8:4::50 eq telnet sequence 10

class-map match-all CLASS-UNOv6
 match access-group name ACLV6-101
class-map match-all CLASS-DOSv6
 match access-group name ACLV6-102
class-map match-all CLASS-TRESv6
 match access-group name ACLV6-103

policy-map POL-OUTv6
 class CLASS-UNOv6
  priority percent 20
 class CLASS-DOSv6
  bandwidth percent 30
    police 8000
 class CLASS-TRESv6
  bandwidth percent 10
 class class-default
  fair-queue
  random-detect

2002(config)# int S2/0
2002(config-if)#
 service-policy output POL-OUTv6
  Policy map POL-OUT is already attached

2002(config-if)#
 no service-policy output POL-OUT
 service-policy output POL-OUTv6
```

C.5. Ejemplo de PRIO en GNU/Linux

En GNU/Linux la herramienta para administrar el tráfico de red es llamada TC (Traffic Conditioner), `tc(8)`. La misma permite hacer policing, shaping, trabajar con varias sub-colas, clasificar/filtrar el tráfico y definir diferentes planificadores de paquetes. En la nomenclatura de TC las políticas de colas se llaman QDISC (Queueing Discipline) y las sub-colas son llamadas clases o bandas. La idea de los ejemplos que se presentan a continuación es mostrar casos muy similares a los que se vieron sobre la maqueta con routers cisco, pero en esta sección sobre una plataforma abierta.

En GNU/Linux, por default, se instala una FIFO de paquetes llamada pfifo_fast (según man page: `tc-pfifo_fast(8)`) en cada interfaz. La cola funciona como una FIFO, pero mide el tamaño de la cola en cantidad paquete y no en bytes. Es una pfifo, `tc-pfifo(8)`, modificada con bandas. A diferencia de como lo hacen estas, también existe la posibilidad de configurar una cola bfifo `tc-bfifo(8)` que cuenta bytes. El modo de descarte para los tres sistemas, pfifo_fast, pfifo y bfifo, es tipo tail drop. En principio son modos de encolados sin clases o categorías, aunque pfifo_fast tiene en realidad 3 colas internas o bandas que se utilizan de acuerdo a un priomap (Priority

Map) definido a partir de la marca de ToS que trae el paquete IP. El formato del ToS y del priomap se verá en el capítulo de DiffServ. Para ver el mecanismo de encolado en las interfaces se pueden ejecutar los siguientes comandos.

```
root@n2# tc qdisc show dev eth1
qdisc pfifo_fast 0: root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

root@n2# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 00:00:00:aa:00:05
          inet addr:11.0.0.2  Bcast:0.0.0.0  Mask:255.255.255.0
          inet6 addr: fe80::200:ff:feaa:5/64 Scope:Link
          inet6 addr: 2001:db8:11::2/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:243 errors:0 dropped:0 overruns:0 frame:0
          TX packets:276 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23078 (23.0 KB)  TX bytes:25968 (25.9 KB)
```

Se podría cambiar por una pfifo sin prioridades.

```
root@n2# tc qdisc add dev eth1 root pfifo limit 100

root@n2# tc qdisc show dev eth1
qdisc pfifo 8008: root refcnt 2 limit 100p
```

Para los ejemplo se usará una disciplina más interesante, una cola classful del tipo PRIO, `tc-prio(8)`, con sub-colas, similar al ejemplo con los routers cisco. De la cola PRIO de GNU/Linux se indica en algunos textos que es como una *pfifo_fast con esteroides*, donde cada banda puede ser una clase separada en lugar de una simple FIFO. Es un sistema de colas classful, lo que significa que permite generar sub-colas de forma jerárquica. De esta manera, sí se podría limitar el tráfico en las colas hijas. En forma pura, sin sub-colas específicas, la cola tipo PRIO es un planificador *Work-Conserving*, es decir, si la interfaz tiene capacidad de enviar tráfico no lo demorará y lo enviará. Trabajando de esta manera, las colas tipo PRIO no hacen shaping y usarán siempre la capacidad disponible del total del enlace. A continuación se muestra la instalación de la cola, las bandas asignadas y la generación de tráfico sin necesidades de QoS.

```
root@n2# tc qdisc del dev eth1 root pfifo limit 100
root@n2# tc qdisc add dev eth1 root handle 1: prio bands 4

root@n2# tc qdisc show dev eth1
qdisc prio 1: root refcnt 2 bands 4
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

root@n2# tc qdisc show
qdisc pfifo_fast 0: dev eth0 root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc prio 1: dev eth1 root refcnt 2 bands 4
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
```

```
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 1408 bytes 16 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:4 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0

root@n2# tc -s filter show dev eth1

root@n5# ping6 -c 10 2001:db8:4::50
PING 2001:db8:4::50(2001:db8:4::50) 56 data bytes
64 bytes from 2001:db8:4::50: icmp_seq=1 ttl=61 time=1.52 ms
64 bytes from 2001:db8:4::50: icmp_seq=2 ttl=61 time=0.187 ms
64 bytes from 2001:db8:4::50: icmp_seq=3 ttl=61 time=0.476 ms
64 bytes from 2001:db8:4::50: icmp_seq=4 ttl=61 time=0.469 ms
64 bytes from 2001:db8:4::50: icmp_seq=5 ttl=61 time=0.483 ms
64 bytes from 2001:db8:4::50: icmp_seq=6 ttl=61 time=0.192 ms
64 bytes from 2001:db8:4::50: icmp_seq=7 ttl=61 time=0.481 ms
64 bytes from 2001:db8:4::50: icmp_seq=8 ttl=61 time=0.197 ms
64 bytes from 2001:db8:4::50: icmp_seq=9 ttl=61 time=0.512 ms
64 bytes from 2001:db8:4::50: icmp_seq=10 ttl=61 time=0.180 ms
--- 2001:db8:4::50 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8998ms
rtt min/avg/max/mdev = 0.180/0.469/1.522/0.378 ms
```

El tráfico se encola según el priomap asignado por default que enviará los paquetes a cada cola de acuerdo al marcado del ToS que estos traigan.

```
root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 6624 bytes 72 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:4 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Para que esto no ocurra se debe cambiar el priomap por uno que ignore las marcas de ToS y enviar el tráfico no indicado específicamente por los filtros deseados a la sub-cola o banda de menor prioridad. Por ejemplo, como se muestra en los siguientes comandos.

```
root@n2# tc qdisc del dev eth1 root handle 1: prio bands 4
root@n2# tc qdisc add dev eth1 root handle 1: prio bands 4
         priomap  3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

root@n2# tc qdisc show dev eth1qdisc prio 1: root refcnt 2 bands 4
          priomap  3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

Se ve ahora, luego del cambio, que el tráfico se encola en la última banda.

```
root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:4 parent 1:
 Sent 176 bytes 2 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Luego se deben colocar los filtros adecuados para encolar según las reglas, de forma similar como se hizo con los routers cisco mediante ACL.

```
root@n2#
#High
tc filter add dev eth1 parent 1: protocol ipv6 prio 1 u32 match
   ip6 dst 2001:DB8:4::1/128
   match ip6 protocol 6 0xff match ip6 dport 23 0xffff flowid 1:1

# Medium
tc filter add dev eth1 parent 1: protocol ipv6 prio 2 u32 match
   ip6 dst 2001:DB8:4::100/128
   match ip6 protocol 58 0xff flowid 1:2

# Normal
tc filter add dev eth1 parent 1: protocol ipv6 prio 3 u32 match
   ip6 dst 2001:DB8:4::50/128
   match ip6 protocol 6 0xff match ip6 dport 23 0xffff flowid 1:3

# Low
tc filter add dev eth1 parent 1: protocol ipv6 prio 4 u32
   match ip6 protocol 17 0xff match ip6 dport 5000 0xffff flowid 1:4
```

Para ver los filtros se puede aplicar el siguiente comando, el cual muestra una salida bastante extensa.

```
root@n2# tc -s filter show dev eth1

filter parent 1: protocol ipv6 pref 1 u32
filter parent 1: protocol ipv6 pref 1 u32 fh 800: ht divisor 1
filter parent 1: protocol ipv6 pref 1 u32 fh 800::800 order 2048
  key ht 800 bkt 0 flowid 1:1
  match 20010db8/ffffffff at 24
  match 00040000/ffffffff at 28
  match 00000000/ffffffff at 32
  match 00000001/ffffffff at 36
  ...
```

Ver de generar nuevamente tráfico de acuerdo a los filtros.

```
root@n5# telnet 2001:db8:4::1
...
root@n5# ping6 -c 10 2001:DB8:4::100
...
root@n5# telnet 2001:db8:4::50
...
```

Se puede ver que se tiene tráfico encolado en cada una de las clases de acuerdo a los filtros instalados.

```
root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 470 bytes 5 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 1180 bytes 10 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 752 bytes 8 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:4 parent 1:
 Sent 16560 bytes 180 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

El mismo efecto se puede lograr marcando internamente el tráfico dentro del kernel con la herramienta `ip6tables(8)` y luego encolándolo según estas marcas con la herramienta `tc(8)`.

```
root@n2# ip6tables -t mangle -F

ip6tables -A POSTROUTING -t mangle -o eth1 -p tcp --dport 23
          -d  2001:DB8:4::1/128 -j MARK --set-mark 1
ip6tables -A POSTROUTING -t mangle -o eth1 -p  ipv6-icmp
          -d  2001:DB8:4::100/128 -j MARK --set-mark 2
ip6tables -A POSTROUTING -t mangle -o eth1 -p tcp --dport 23
          -d  2001:DB8:4::50/128 -j MARK --set-mark 3
ip6tables -A POSTROUTING -t mangle -o eth1 -p udp --dport 5000
          -j MARK --set-mark 4

root@n2# ip6tables -t mangle -L -n -v
...
Chain POSTROUTING (policy ACCEPT 2 packets, 160 bytes)
 pkts bytes target     prot opt in     out     source
 destination
    0     0 MARK       tcp      *      eth1    ::/0
 2001:db8:4::1/128    tcp dpt:23 MARK set 0x1
    0     0 MARK       icmpv6    *      eth1   ::/0
 2001:db8:4::100/128  MARK set 0x2
    0     0 MARK       tcp      *      eth1    ::/0
 2001:db8:4::50/128   tcp dpt:23 MARK set 0x3
    0     0 MARK       udp      *      eth1    ::/0
 ::/ 0                udp dpt:5000 MARK set 0x4
```

Luego de la reglas de marcado se deben aplicar los filtros en relación a las marcas internas previamente configuradas.

```
root@n2#
tc filter add dev eth1 parent 1: protocol ipv6 handle 1 fw flowid 1:1
tc filter add dev eth1 parent 1: protocol ipv6 handle 2 fw flowid 1:2
tc filter add dev eth1 parent 1: protocol ipv6 handle 3 fw flowid 1:3
tc filter add dev eth1 parent 1: protocol ipv6 handle 4 fw flowid 1:4

root@n2# tc -s filter show dev eth1
filter parent 1: protocol ipv6 pref 49149 fw
filter parent 1: protocol ipv6 pref 49149 fw handle 0x4 classid 1:4
filter parent 1: protocol ipv6 pref 49150 fw
filter parent 1: protocol ipv6 pref 49150 fw handle 0x3 classid 1:3
filter parent 1: protocol ipv6 pref 49151 fw
filter parent 1: protocol ipv6 pref 49151 fw handle 0x2 classid 1:2
filter parent 1: protocol ipv6 pref 49152 fw
filter parent 1: protocol ipv6 pref 49152 fw handle 0x1 classid 1:1
```

Dentro de las clases de cada cola PRIO se maneja el tráfico como una cola "PACKET FIFO", p_fifo. Se puede cambiar este funcionamiento a una cola tipo WFQ, en

términos de Linux, SFQ (Stochastic Fairness Queueing), `tc-sfq(8)` o un TBF (Token Bucket Filter), `tc-tbf(8)`, que limita el tráfico haciendo shaping. A continuación el ejemplo.

```
root@n2#
tc qdisc add dev eth1 parent 1:1 handle 10: sfq
tc qdisc add dev eth1 parent 1:2 handle 20: tbf rate 10kbit buffer 1600
   limit 3000
tc qdisc add dev eth1 parent 1:3 handle 30: sfq

root@n2# tc class show dev eth1
class prio 1:1 parent 1: leaf 10:
class prio 1:2 parent 1: leaf 20:
class prio 1:3 parent 1: leaf 30:
class prio 1:4 parent 1:
class tbf 20:1 parent 20:

root@n2# tc -d qdisc show dev eth1
qdisc prio 1: root refcnt 2 bands 4
      priomap  3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
qdisc sfq 10: parent 1:1 limit 127p quantum 1514b
      flows 127/1024 divisor 1024
qdisc tbf 20: parent 1:2 rate 10000bit burst 1600b/8
      mpu 0b lat 1.1s
qdisc sfq 30: parent 1:3 limit 127p quantum 1514b
      flows 127/1024 divisor 1024
```

Las colas de tipo TBF son consideradas *non-Work-Conserving*, es decir, pueden limitar el tráfico mediante shaping aún habiendo capacidad en la interfaz para enviar los paquetes. Si se prueba el ejemplo se ve que el tráfico se demora y descarta cuando se llena la cola de acuerdo a los parámetros configurados. En este caso una capacidad de 10Kbps.

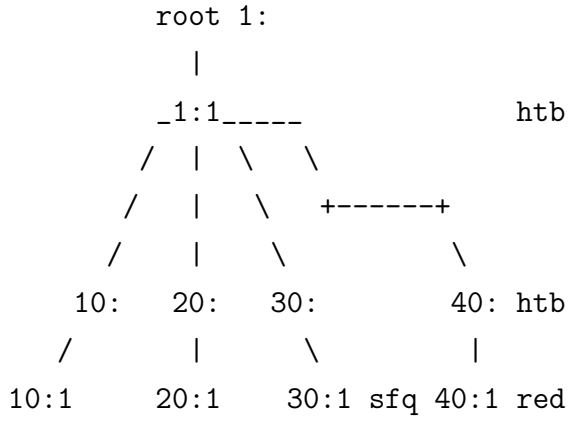
```
root@n5# ping6 2001:db8:4::100 -A -s 1500
PING 2001:db8:4::100(2001:db8:4::100) 1500 data bytes
1508 bytes from 2001:db8:4::100: icmp_seq=1 ttl=61 time=33.4 ms
1508 bytes from 2001:db8:4::100: icmp_seq=2 ttl=61 time=1310 ms
^C
--- 2001:db8:4::100 ping statistics ---
53 packets transmitted, 2 received, 96% packet loss, time 3853ms
rtt min/avg/max/mdev = 33.457/672.195/1310.934/638.739 ms, pipe 39,
ipg/ewma 74.099/193.141 ms

root@n2# tc -s class show dev eth1
class prio 1:1 parent 1: leaf 10:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1: leaf 20:
 Sent 9212 bytes 30 pkt (dropped 76, overlimits 140 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1: leaf 30:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:4 parent 1:
 Sent 16844 bytes 192 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class tbf 20:1 parent 20:
```

Las colas de tipo SFQ que se usaron como banda para el tráfico de telnet son colas que funcionan como las WFQ, clasificando de acuerdo a flujos y distribuyendo

los paquetes en una gran cantidad de colas internas. SFQ es classless, o sea no permite crear otras colas o bandas dentro de las definidas de forma dinámica para cada flujo, además SFQ es una disciplina *Work-conservative*, debido a que siempre aprovechará la capacidad de la interfaz. En el ejemplo sería conveniente a la cola 1:4 también asignarle una SFQ en lugar de dejarla con una pfifo. que es la banda default para las sub-colas de PRIO.

C.6. Ejemplo de HTB/CBQ en GNU/Linux

El ejemplo anterior se vio sobre una cola PRIO. Este tipo de cola tiene la limitación de que si el tráfico de mayor prioridad es grande y no se lo limita, puede causar starvation, inanición, del resto del tráfico. En el siguiente ejemplo se mostrará el uso de una política tipo CBQ. En GNU/Linux las colas CBQ son bastante complejas de configurar y, en ocasiones, no parecen ser apropiadas para algunas situaciones. En lugar de utilizar una CBQ se mostrará el ejemplo con una HTB. La clase HTB (Hierarchy Token Bucket) es un reemplazo de CBQ más sencillo e intuitivo. Tanto CBQ como HTB permiten controlar el tráfico saliente haciendo shaping y/o policing. Ambos permiten crear sub-colas y mandar el tráfico a estas de acuerdo a filtros. CBQ requiere saber de ante mano el ancho de banda total de la interfaz, HTB no lo requiere y hace el control con TBF. Al permitir shaping son *Non-work-conservative*. A continuación de muestra un diagrama en ASCII Art de la estructura y de forma seguida los comandos para generarla.



Para la configuración, primero se suprime la estructura de disciplina anterior.

```
root@n2# tc qdisc del dev eth1 root handle 1: prio bands 4

root@n2# tc qdisc show
qdisc pfifo_fast 0: dev eth0 root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

```
qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

root@n2# tc filter show
```

Posteriormente se aplican los comandos para instalar la HTB.

```
root@n2#
tc qdisc add dev eth1 root handle 1: htb default 40

tc class add dev eth1 parent 1: classid 1:1 htb rate 10mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:10 htb rate 7mbit
   ceil 10mbps burst 15k
tc class add dev eth1 parent 1:1 classid 1:20 htb rate 2mbit
   ceil 4mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:30 htb rate 200kbit
   ceil 2mbit burst 15k
tc class add dev eth1 parent 1:1 classid 1:40 htb rate 800kbit

tc qdisc add dev eth1 parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev eth1 parent 1:20 handle 20: sfq perturb 10
tc qdisc add dev eth1 parent 1:30 handle 30: sfq perturb 10
```

Para usar RED como política de descarte se requiere indicar de forma explícita los parámetros en la configuración, todos expresados en bytes, salvo burst que se da en paquetes. En el caso de cisco los calculaba de forma automática.

```
Usage: ...red limit BYTES min BYTES max BYTES avpkt BYTES burst PACKETS
              probability PROBABILITY bandwidth KBPS [ ecn ]
```

Si no se colocan de forma adecuada da un error.

```
RED: failed to calculate EWMA constant
```

En este caso indica que no puede calcular el *Exponentially Weighted Moving Averages*. Para calcular los parámetros se pueden usar las siguientes reglas.

Un valor necesario es Min. En este está reflejado el valor máximo de delay aceptable en cola, que se debe luego multiplicar por el ancho de banda. Si se pone un valor muy chico degrada el rendimiento, y, si es grande puede agregar demasiado delay. Por ejemplo, podría calcularse con base de 800Kbps de ancho de banda digital y 200ms de delay, dando 20000 bytes.

$$BW = 800Kbps, Delay = 200ms, Min = \frac{200 \times 800)}{8} = 20000bytes \tag{C.1}$$

Max se puede colocar como 2 veces el Min. Este parámetro previene la sincronización TCP. Si el link es de poco bandwidth y se pierde tráfico, cambiar por 4 veces el Min.

$$Max = 2 \times Min = 2 = 20000 = 40000bytes \tag{C.2}$$

Limit es el máximo tamaño de la cola si fuese tail-drop. Un posible valor es 8 veces el Max. *Avpkt* es el promedio de bytes de paquetes en cola. 1000 indica 1 paquete o menos si se toma un MTU=1500bytes.

$$Limit = 8 \times Max = 320000bytes, Avpkt = 1000bytes \tag{C.3}$$

Burst indica la cantidad máxima de paquetes que pasarán de una ráfaga. Este valor controla como RED responde a las ráfagas. Burst debe ser mayor a min/avpkt. Experimentalmente un valor adecuado puede ser:

$$Burst = \frac{(Min + Min + Max)}{(3 \times avpkt)} \tag{C.4}$$

$$Burst = \frac{(2000 + 2000 + 4000)}{3 \times avpkt} = \frac{(2000 + 2000 + 4000)}{3 \times 1000} = 2pkts \tag{C.5}$$

Finalmente, en nuestro ejemplo el comando para activar RED será el siguiente.

```
tc qdisc add dev eth1 parent 1:40 handle 40: red limit 32000 min 20000
   max 40000 avpkt 1000 burst 40 bandwidth 800kbit ecn
```

Por último, la configuración establecida es la que se muestra.

```
root@n2# tc qdisc show dev eth1
qdisc htb 1: root refcnt 2 r2q 10 default 40 direct_packets_stat 0
qdisc sfq 10: parent 1:10 limit 127p quantum 1514b divisor 1024
      perturb 10sec
qdisc sfq 20: parent 1:20 limit 127p quantum 1514b divisor 1024
      perturb 10sec
qdisc sfq 30: parent 1:30 limit 127p quantum 1514b divisor 1024
      perturb 10sec
qdisc red 40: parent 1:40 limit 32000b min 20000b max 40000b ecn

root@n2# tc class show dev eth1
class htb 1:1 root rate 10000Kbit ceil 10000Kbit burst 15Kb cburst 1600b
class htb 1:10 parent 1:1 leaf 10: prio 0 rate 7000Kbit ceil 80000Kbit
 burst 15Kb cburst 1600b
 class htb 1:20 parent 1:1 leaf 20: prio 0 rate 2000Kbit ceil 4000Kbit
 burst 15Kb cburst 1600b
class htb 1:30 parent 1:1 leaf 30: prio 0 rate 200000bit ceil 2000Kbit
 burst 15Kb cburst 1600b
class htb 1:40 parent 1:1 leaf 40: prio 0 rate 800000bit ceil 800000bit
 burst 1599b cburst 1599b
class red 40:1 parent 40:
```

Y los filtros aplicados

```
root@n2#
## CLASE-UNO
tc filter add dev eth1 parent 1: protocol ipv6 prio 1 u32 match
   ip6 dst 2001:DB8:4::1/128
   match ip6 protocol 6 0xff match ip6 dport 5001 0xffff flowid 1:10
## CLASE-DOS
tc filter add dev eth1 parent 1: protocol ipv6 prio 2 u32
   ip6 dst 2001:DB8:4::100/128
   match ip6 protocol 58 0xff flowid 1:20
## CLASE-TRES
tc filter add dev eth1 parent 1: protocol ipv6 prio 3 u32 match
   ip6 dst 2001:DB8:4::50/128
   match ip6 protocol 6 0xff match ip6 dport 5002 0xffff flowid 1:30
## CLASE-DEFAULT
tc filter add dev eth1 parent 1: protocol ipv6 prio 4 u32
   match ip6 protocol 17 0xff match ip6 dport 5000 0xffff flowid 1:40
```

Seguidamente se muestra el estado generado por la configuración.

```
root@n2# tc -s class show dev eth1
class htb 1:1 root rate 10000Kbit ceil 10000Kbit burst 15Kb cburst 1600b
 Sent 3114 bytes 35 pkt (dropped 0, overlimits 0 requeues 0)
 rate 160bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 190813 ctokens: 18813

class htb 1:10 parent 1:1 leaf 10: prio 0 rate 7000Kbit ceil 80000Kbit
 burst 15Kb cburst 1600b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 274281 ctokens: 2500

class htb 1:20 parent 1:1 leaf 20: prio 0 rate 2000Kbit ceil 4000Kbit
 burst 15Kb cburst 1600b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 960000 ctokens: 50000

class htb 1:30 parent 1:1 leaf 30: prio 0 rate 200000bit ceil 2000Kbit
 burst 15Kb cburst 1600b
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 rate 0bit 0pps backlog 0b 0p requeues 0
 lended: 0 borrowed: 0 giants: 0
 tokens: 9600000 ctokens: 100000

class htb 1:40 parent 1:1 leaf 40: prio 0 rate 800000bit ceil 800000bit
 burst 1599b cburst 1599b
 Sent 3114 bytes 35 pkt (dropped 0, overlimits 0 requeues 0)
 rate 160bit 0pps backlog 0b 0p requeues 0
 lended: 35 borrowed: 0 giants: 0
 tokens: 234984 ctokens: 234984

class red 40:1 parent 40:
```

Para verla en acción se puede probar con el comando `iperf(8)` generando tráfico TCP con las características que hacen matching con las reglas escritas.

```
root@n5# iperf -V -c 2001:DB8:4::1 -p 5001
------------------------------------------------------------
Client connecting to 2001:DB8:4::1, TCP port 5001
TCP window size: 20.8 KByte (default)
------------------------------------------------------------
[  3] local 2001:db8:20::100 port 56390 connected with 2001:db8:4::1
port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.1 sec  11.4 MBytes  9.41 Mbits/sec

root@n2# tc -s class show dev eth1 | grep Sent
Sent 18624700 bytes 12362 pkt (dropped 0, overlimits 0 requeues 0)
Sent 18804460 bytes 12428 pkt (dropped 14,overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 4948 bytes 56 pkt (dropped 0, overlimits 0 requeues 0)

root@n5# iperf -V -c 2001:DB8:4::50 -p 5002
------------------------------------------------------------
Client connecting to 2001:DB8:4::50, TCP port 5002
TCP window size: 20.8 KByte (default)
------------------------------------------------------------
[  3] local 2001:db8:20::100 port 41415 connected with 2001:db8:4::50
port 5002
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.7 sec  2.62 MBytes  2.06 Mbits/sec

root@n2# tc -s class show dev eth1 | grep Sent
```

```
 Sent 28115158 bytes 18665 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 25113186 bytes 16597 pkt (dropped 16, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
Sent 2994396 bytes 1982 pkt (dropped 31, overlimits 0 requeues 0)
 Sent 7576 bytes 86 pkt (dropped 0, overlimits 0 requeues 0)

root@n5# ping6 -f -c 10000 -s 1500 2001:DB8:4::1 &
         ping6 -f -c 10000 -s 1500 2001:DB8:4::1

root@n2#
...
class htb 1:40 parent 1:1 leaf 40: prio 0 rate 800000bit
      ceil 800000bit burst 1599b cburst 1599b
Sent 6111378 bytes 7829 pkt (dropped 1021, overlimits 0 requeues 0)
 rate 555488bit 91pps backlog 0b 54p requeues 0
 lended: 7775 borrowed: 0 giants: 0
 tokens: -233304 ctokens: -233304
```

Para probar RED en combinación con ECN (Explicit Congestion Notification) se debe activar en ambos extremos, emisor y receptor, la capacidad de trabajar con ECN.

```
root@n4# echo 1 > /proc/sys/net/ipv4/tcp_ecn &&
cat /proc/sys/net/ipv4/tcp_ecn
1
root@n5# echo 1 > /proc/sys/net/ipv4/tcp_ecn &&
cat /proc/sys/net/ipv4/tcp_ecn
1
```

Siguiendo con los tests, ver la limitación del ancho de banda asignado.

```
root@n5# iperf -V -c 2001:DB8:4::1 -p 5003
------------------------------------------------------------
Client connecting to 2001:DB8:4::1, TCP port 5003
TCP window size: 20.8 KByte (default)
------------------------------------------------------------
[  3] local 2001:db8:20::100 port 36662 connected with 2001:db8:4::1
      port 5003
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-11.9 sec  1.12 MBytes   796 Kbits/sec
```

El tráfico se regula con RED marcando la congestión desde la red (los routers) a los extremos que soportan el método de congestión explícita notificada por la red. Los routers la detectan de acuerdo a los flags de ECN en los datagramas IP. En este caso, la reacción del emisor será como si el segmento se hubiese descartado entrando en el control de congestión TCP, aunque el router no lo descartará, sino lo usará para notificar del evento. La regulación con ECN está definida en el RFC-3168 [RFB01]. En el mismo documento se indican los flags que se utilizan en los segmentos TCP y en los datagramas IP. Para ambos casos son 2 indicadores que forman parte de los paquetes marcados sobre "espacios" no utilizados. Los campos del datagrama IP se ven en el capítulo que trata el modelo de QoS de DiffServ. Para el caso de TCP los flags son los mostrados en la figura C.2.

La negociación del control explícito de la congestión por la red soportada por los extremos de la sesión TCP se realiza durante el inicio de la conexión, **three-way handshake**. De forma esquemática se muestra en la figura C.3 el intercambio de

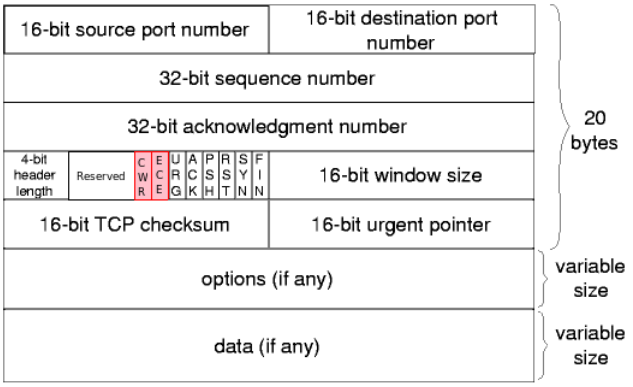


Figura C.2: Flags de TCP para indicar congestión por parte de la red

mensajes donde se ve como se activan los flags. Una vez negociado a nivel de capa de transporte, todos los paquetes IP transmitidos durante la sesión TCP deben llevar los indicadores, flags, para que los routers puedan discriminar si deben tratar los datos con ECN notificando al receptor, o de forma tradicional, directamente descartando.

El mensaje de SYN (ECN-Setup) para el establecimiento de la conexión se muestra en la figura C.4 luego en la figura C.5 se muestra como se marcan los datagramas IP que llevan datos para indicar que el protocolo de transporte encapsulado sobre IP soporta ECN de acuerdo al flag ECN-Capable.

En la figura C.6 se muestra como se marcan los datagramas IP por los routers con soporte de ECN al detectar la congestión real o inminente de acuerdo a RED. De esta forma avisan al receptor que la misma se está produciendo.

El receptor debe notificar el evento al emisor y este tiene que tomar medidas reduciendo la ventana de congestión, como si se hubiese producido el descarte. En la figura C.7 se muestra el intercambio de flags TCP para indicar congestión: ECN-Echo, desde el receptor al emisor. Después que la misma ha sido tenida en cuenta por el origen de los datos se anuncia mediante la confirmación con CWR (Congestion Window Reduced).

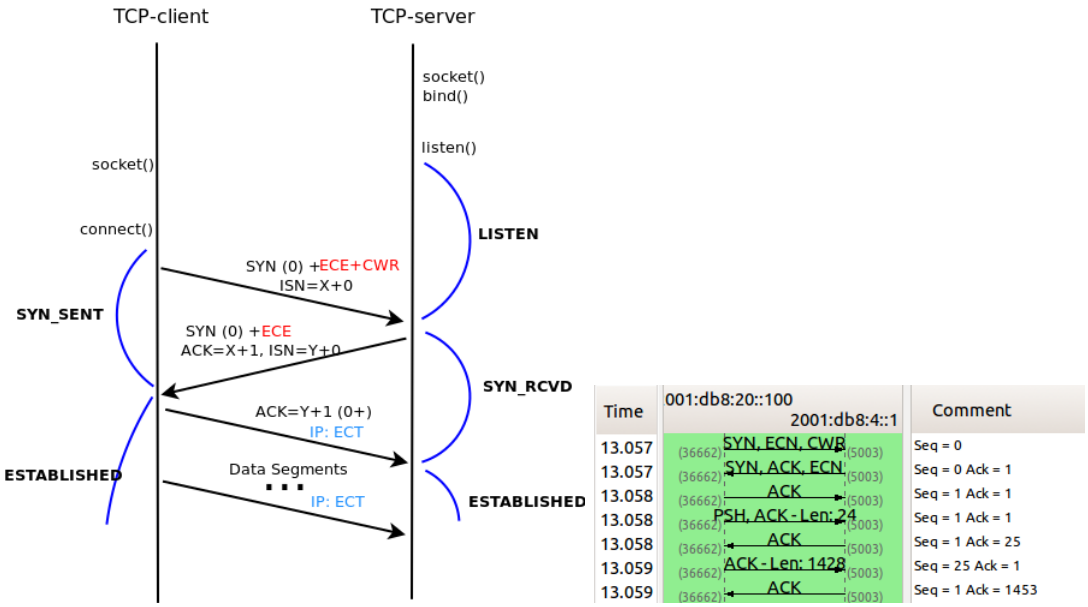


Figura C.3: Negociación de ECN durante la conexión TCP

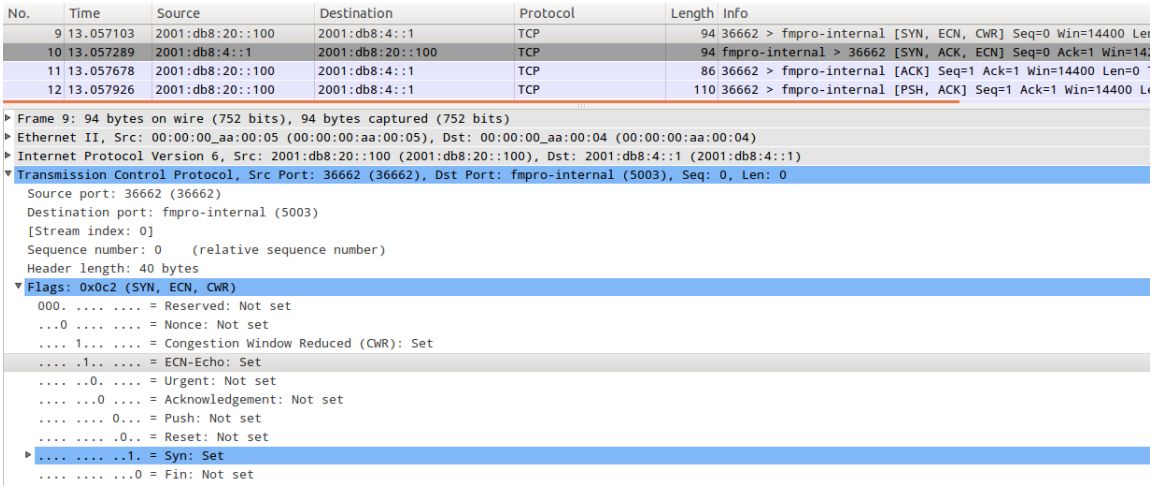


Figura C.4: Mensaje de ECN-Setup de la sesión TCP

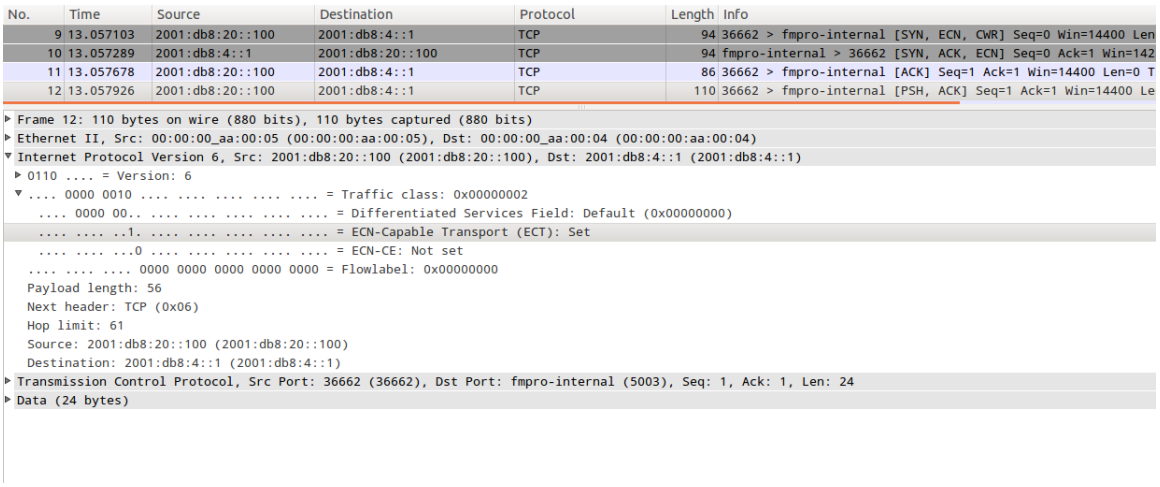


Figura C.5: Datagrama IP con el flag de soporte de ECN-Capable

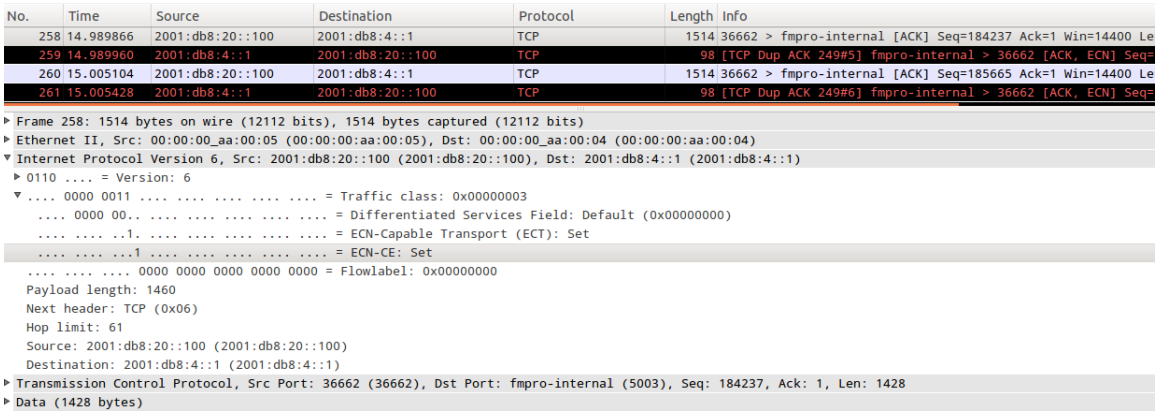


Figura C.6: Datagrama IP con el flag de soporte de ECN-CE

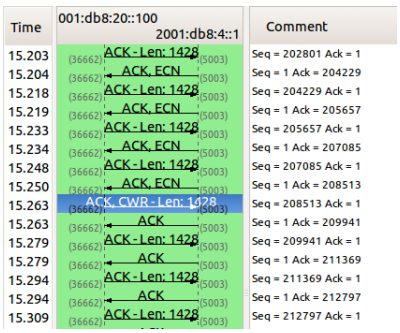


Figura C.7: Intercambio de Flags TCP para indicar congestión

C.7. Ejemplo de Ring Buffers

A continuación otro ejemplo relacionado con el manejo de colas. En esta sección se trata con buffers de bajo nivel. Para ilustrar se presentan comandos que permiten acceder a ver el estado de los ring buffers, TX y RX-rings en el caso de las plataformas cisco y GNU/Linux.

C.7.1. Ejemplo de Ring Buffers en Cisco

El ring en las interfaces de los equipos cisco se puede ver con el siguiente comando.

```
#show controllers Serial 0/0/0
...
Receive Ring
rxr head (0)(0x0F644520), rxr tail (0)(0x0F644520)
rmd(F644520):nbd F644530 cmd_sts 80800000 buf_sz 06000000 buf_ptr F6468E0
rmd(F644530):nbd F644540 cmd_sts 80800000 buf_sz 06000000 buf_ptr F646F40
...
Transmit Ring
txr head (0)(0x0F646040), txr tail (0)(0x0F646040)
tmd(F646040): nbd F646050 cmd_sts 00C30000 byt_cnt ABCDABCD buf_ptr 0
tmd(F646050): nbd F646060 cmd_sts 00C30000 byt_cnt ABCDABCD buf_ptr 0
...
```

En el caso de GNU/Linux también se pueden observar el TX-ring y el RX-ring con el comando `ethtool(8)`.

```
# ethtool --show-ring eth0
 Ring parameters for eth0:
 Pre-set maximums:
 RX: 4096
 RX Mini: 0
 RX Jumbo: 0
 TX: 4096
 Current hardware settings:
 RX: 256
 RX Mini: 0
 RX Jumbo: 0
 TX: 256
```

Se podrían hacer varias pruebas más, pero el texto quedaría demasiado largo. En el apéndice se alcanzaron a mostrar diversos ejemplo que sirven a modo ilustrativo para poder comprender los sistemas de encolado y el manejo de buffers en equipos reales.

Apéndice D

Ejemplos con DiffServ

En este apéndice se mostrarán algunos casos de uso y configuraciones de ejemplo para brindar QoS usando el modelo DiffServ. Los ejemplos se mostrarán sobre la plataforma de router cisco y sobre GNU/Linux. Todos los flujos serán generados de acuerdo a la figura D.1 que es la misma topología usada para los ejemplos de encolado. El router con ID **2001** cumple el rol de router de borde, edge del dominio, tomando la responsabilidad del marcado y acondicionamiento y los routers **2002** y **2004** cumplen los roles de routers de core. En ese caso el destino del tráfico está dentro del dominio de QoS DS.

D.1. Ejemplo DiffServ sobre cisco IPv4

En los nodos de borde, en este caso el router 2001, la configuración es la siguiente.

```
2001(config)#
 access-list 101 permit tcp any host 4.0.0.1 eq telnet
 access-list 102 permit icmp any host 4.0.0.100
 access-list 103 permit tcp any host 4.0.0.50 eq telnet

 class-map match-all CLASS-UNO
  match access-group 101
```

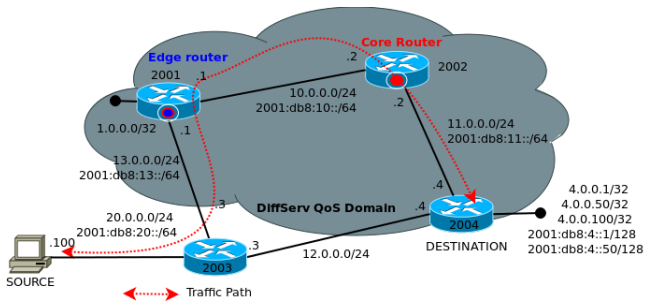


Figura D.1: Topología de prueba para ver ejemplos con el modelo DiffServ

```
class-map match-all CLASS-DOS
 match access-group 102
class-map match-all CLASS-TRES
 match access-group 103

policy-map POL-DIFFSERV
 class CLASS-UNO
    police 20000 conform-action set-dscp-transmit ef
    exceed-action dro
 class CLASS-DOS
    police 400000 conform-action set-dscp-transmit af12
                  exceed-action set-dscp-transmit default
 class CLASS-TRES
    police 800000 conform-action set-dscp-transmit af22
                  exceed-action set-dscp-transmit default
 class class-default
  fair-queue
  random-detect
```

En la misma se muestra la ACL para hacer matching con el tráfico a marcar. Sirve, también, para aplicar el descarte del exceso para el caso del tráfico más prioritario o el remarcado como "best-effort" para las dos siguientes clases. Se usa el marcado de Expedited Forwarding (EF) para el tráfico de mayor prioridad y dos clases de Assured Forwarding para las restantes. Finalmente se coloca una WFQ con RED. Seguidamente se aplica. Se muestra que al estar en un CBWFQ solo se puede colocar en la interfaz de salida, aunque sería más natural aplicarlo a la entrada.

```
2001(config)#interface Serial2/1
2001(config-if)#service-policy input POL-DIFFSERV
CBWFQ : Can be enabled as an output feature only

interface Serial2/0
 description ####_2002.S2/0_####
 ip address 10.0.0.1 255.255.255.0
 ipv6 address 2001:DB8:10::1/64
 ipv6 ospf 1 area 0.0.0.0
 serial restart-delay 0
 service-policy output POL-DIFFSE
end
```

Luego se presenta la configuración en los nodos de core donde se aplican las políticas de acuerdo al tráfico marcado en los bordes. A cada cola se le asegura un ancho de banda y se coloca el tráfico marcado con EF como el prioritario.

```
2002(config)#
 class-map match-all CLASS-UNO-DS
  match  dscp ef
 class-map match-all CLASS-DOS-DS
  match  dscp af12
 class-map match-all CLASS-TRES-DS
  match  dscp af22

 policy-map POL-DIFSERV-OUT
  class CLASS-UNO-DS
   priority percent 20
  class CLASS-DOS-DS
   bandwidth percent 30
  class CLASS-TRES-DS
   bandwidth percent 10
  class class-default
   fair-queue
  random-detect
```

También debe aplicarse en las interfaces.

```
2002(config)#
 interface Serial2/1
  description ####_2004.Serial2/1_####
  ip address 11.0.0.2 255.255.255.0
  load-interval 30
  ipv6 address 2001:DB8:11::2/64
  ipv6 ospf 1 area 0.0.0.0
  serial restart-delay 0
  service-policy output POL-DIFSERV-OUT
 end
```

En el siguiente paso se configura el nodo que generará el tráfico.

```
root@source:~#

 ifconfig tap0 20.0.0.100 netmask 255.255.255.0 up
 route add -host 4.0.0.1 gw 20.0.0.3
 route add -host 4.0.0.50 gw 20.0.0.3
 route add -host 4.0.0.100 gw 20.0.0.3

 route add -net 11.0.0.0 netmask 255.255.255.0 gw 20.0.0.3
 route add -net 10.0.0.0 netmask 255.255.255.0 gw 20.0.0.3
 route add -net 13.0.0.0 netmask 255.255.255.0 gw 20.0.0.3

root@source:~# traceroute -n -I 4.0.0.1
traceroute to 4.0.0.1 (4.0.0.1), 30 hops max, 60 byte packets
 1  20.0.0.3  3.171 ms  3.373 ms  3.558 ms
 2  13.0.0.1  10.420 ms  10.770 ms  10.869 ms
 3  10.0.0.2  17.297 ms  21.783 ms  21.963 ms
 4  4.0.0.1  26.298 ms  26.458 ms  32.976 ms
```

Posteriormente se prueba como se marca el tráfico de acuerdo a las ACL colocadas en los nodos de borde.

```
root@source:~# telnet 4.0.0.1
Trying 4.0.0.1...
Connected to 4.0.0.1.
Escape character is '^]'.
```

En la imagen D.2 se ve como el origen genera el paquete IP donde se transporta el mensaje de telnet que es marcado según RFC-791 con 0x04, lo que significa que minimize el delay (MD). Esta captura se produce en el enlace entre los routers 2003 y 2001. En el borde se ve:

```
2001#show access-lists
Extended IP access list 101
    10 permit tcp any host 4.0.0.1 eq telnet (36 matches)
Extended IP access list 102
    10 permit icmp any host 4.0.0.100
Extended IP access list 103
    10 permit tcp any host 4.0.0.50 eq telnet


2001#show policy-map interface Serial 2/0
 Serial2/0

  Service-policy output: POL-DIFFSERV

    Class-map: CLASS-UNO (match-all)
      36 packets, 1684 bytes
```

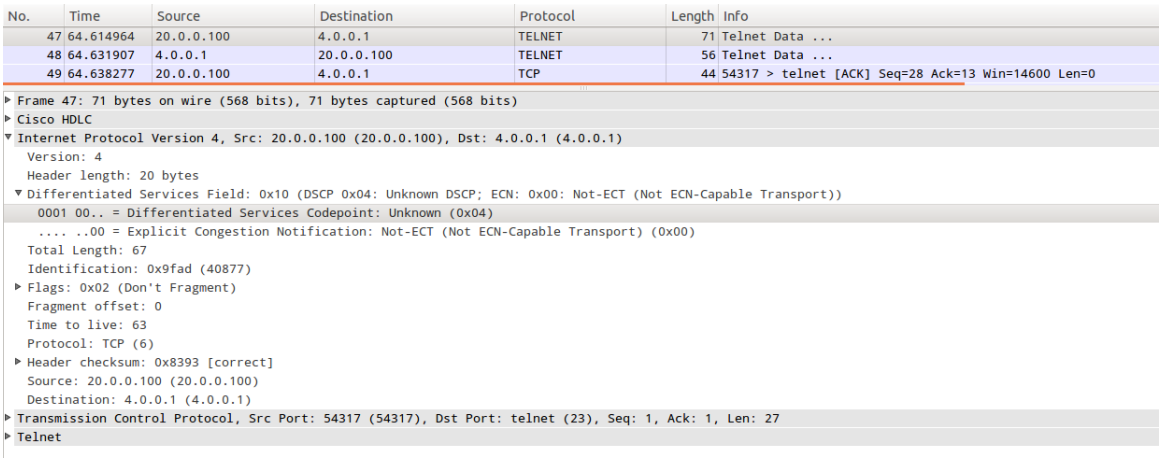


Figura D.2: Mensaje de telnet marcado desde el origen con ToS=MD

```
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: access-group 101
    police:
        cir 20000 bps, bc 1500 bytes
      conformed 36 packets, 1684 bytes; actions:
        set-dscp-transmit ef
      exceeded 0 packets, 0 bytes; actions:
        drop
      conformed 0 bps, exceed 0 bps

  Class-map: CLASS-DOS (match-all)
    0 packets, 0 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: access-group 102
    police:
        cir 400000 bps, bc 12500 bytes
      conformed 0 packets, 0 bytes; actions:
        set-dscp-transmit af12
      exceeded 0 packets, 0 bytes; actions:
        set-dscp-transmit default
      conformed 0 bps, exceed 0 bps

  Class-map: CLASS-TRES (match-all)
    0 packets, 0 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: access-group 103
    police:
        cir 800000 bps, bc 25000 bytes
      conformed 0 packets, 0 bytes; actions:
        set-dscp-transmit af22
      exceeded 0 packets, 0 bytes; actions:
        set-dscp-transmit default
      conformed 0 bps, exceed 0 bps

  Class-map: class-default (match-any)
    317 packets, 24728 bytes
    5 minute offered rate 0 bps, drop rate 0 bps
    Match: any
    Queueing
      Flow Based Fair Queueing
      Maximum Number of Hashed Queues 256
      (total queued/total drops/no-buffer drops) 0/0/0
       exponential weight: 9
```

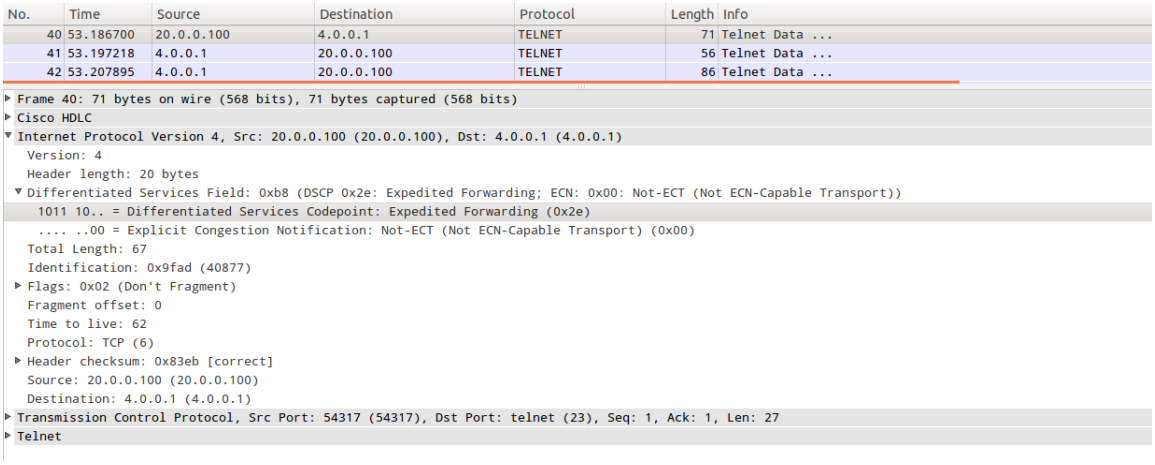


Figura D.3: Mensaje de telnet marcado desde el origen con DSCP=EF

...

En la figura D.3 se ve como el paquete IP es remarcado en el borde, ignorando la marca original 0x04 y se coloca la marca de EF (Expedited Forwarding), 0x2E. Esta captura es tomada entre los routers 2001 y 2002. Se ve en las interfaces de los nodos de core el matching.

```
2002#show policy-map interface Serial 2/1
 Serial2/1

  Service-policy output: POL-DIFSERV-OUT

    Class-map: CLASS-UNO-DS (match-all)
      36 packets, 1684 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match:  dscp ef (46)
      Queueing
        Strict Priority
        Output Queue: Conversation 264
        Bandwidth 20 (%)
        Bandwidth 308 (kbps) Burst 7700 (Bytes)
        (pkts matched/bytes matched) 0/0
        (total drops/bytes drops) 0/0

    Class-map: CLASS-DOS-DS (match-all)
      0 packets, 0 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match:  dscp af12 (12)
      Queueing
        Output Queue: Conversation 265
        Bandwidth 30 (%)
        Bandwidth 463 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: CLASS-TRES-DS (match-all)
      0 packets, 0 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match:  dscp af22 (20)
```

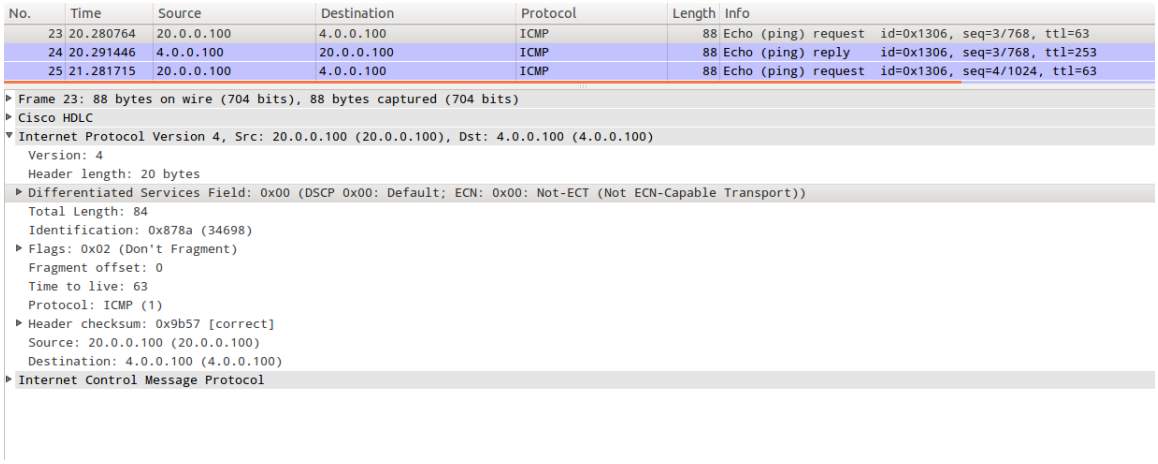


Figura D.4: Mensaje ICMP original

```
      Queueing
        Output Queue: Conversation 266
        Bandwidth 10 (%)
        Bandwidth 154 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: class-default (match-any)
      220 packets, 18156 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match: any
      Queueing
        Flow Based Fair Queueing
        Maximum Number of Hashed Queues 256
        (total queued/total drops/no-buffer drops) 0/0/0
         exponential weight: 9
...
```

Prontamente se muestra como generar tráfico de las otras clases definidas y remarcadas con AF.

```
root@source:~# ping 4.0.0.100 -c 20
PING 4.0.0.100 (4.0.0.100) 56(84) bytes of data.
64 bytes from 4.0.0.100: icmp_req=1 ttl=252 time=18.1 ms
64 bytes from 4.0.0.100: icmp_req=2 ttl=252 time=19.3 ms
64 bytes from 4.0.0.100: icmp_req=3 ttl=252 time=17.4 ms
64 bytes from 4.0.0.100: icmp_req=4 ttl=252 time=16.7 ms
64 bytes from 4.0.0.100: icmp_req=5 ttl=252 time=18.0 ms
...
--- 4.0.0.100 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19026ms
rtt min/avg/max/mdev = 14.325/18.141/23.636/2.537 ms
```

En la figura D.4 se ve como el paquete IP que lleva ICMP es marcado con "best-effort" (0x00) por el origen. La captura es tomada entre los routers 2003 y 2001.

En el borde se ve el matching.

```
2001#show access-lists
Extended IP access list 101
    10 permit tcp any host 4.0.0.1 eq telnet (55 matches)
Extended IP access list 102
```

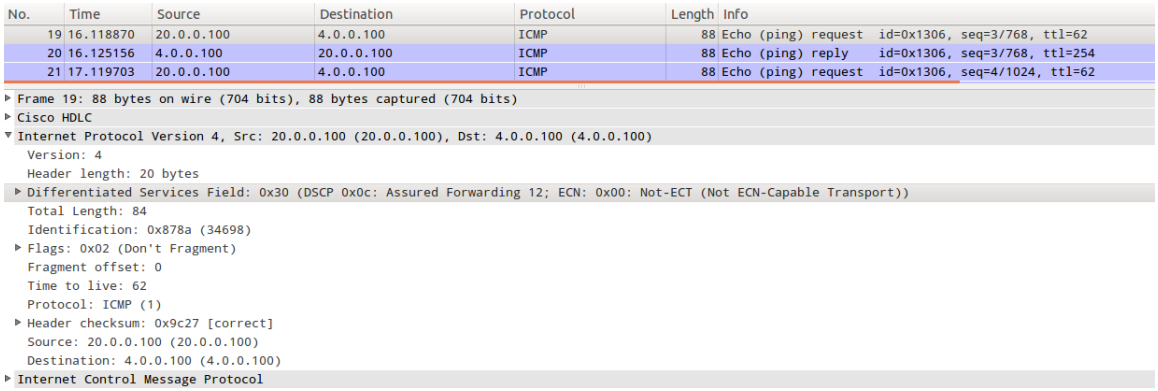


Figura D.5: Mensaje ICMP re-marcado con DSCP=AF12

```
    10 permit icmp any host 4.0.0.100 (20 matches)
Extended IP access list 103
    10 permit tcp any host 4.0.0.50 eq telnet
2001#
2001#show policy-map interface Serial 2/0
 Serial2/0

  Service-policy output: POL-DIFFSERV

    ...

    Class-map: CLASS-DOS (match-all)
      20 packets, 1760 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 102
      police:
          cir 400000 bps, bc 12500 bytes
        conformed 20 packets, 1760 bytes; actions:
          set-dscp-transmit af12
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit default
        conformed 0 bps, exceed 0 bps

    Class-map: CLASS-TRES (match-all)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 103
      police:
          cir 800000 bps, bc 25000 bytes
        conformed 0 packets, 0 bytes; actions:
          set-dscp-transmit af22
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit default
        conformed 0 bps, exceed 0 bps
        ...
```

En la figura D.5 se ve como el paquete IP con ICMP que tenía la marca de "best-effort", 0x00, es remarcado por AF12 (Assured Forwarding), 0x0C. La captura es tomada entre los routers 2001 y 2002. En el core se ve el matching de la marca AF12.

```
2002#show policy-map interface Serial 2/1
```

```
 Serial2/1

  Service-policy output: POL-DIFSERV-OUT
    ...
    Class-map: CLASS-DOS-DS (match-all)
      20 packets, 1760 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match:  dscp af12 (12)
      Queueing
        Output Queue: Conversation 265
        Bandwidth 30 (%)
        Bandwidth 463 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0
     ...
```

Para la otra clase también se realizan las pruebas y se ve el matching.

```
root@source:~# telnet 4.0.0.50
Trying 4.0.0.50...
Connected to 4.0.0.50.

2001#show access-lists
Extended IP access list 101
    10 permit tcp any host 4.0.0.1 eq telnet (55 matches)
Extended IP access list 102
    10 permit icmp any host 4.0.0.100 (313 matches)
Extended IP access list 103
    10 permit tcp any host 4.0.0.50 eq telnet (33 matches)

2001#show policy-map interface Serial 2/0
 Serial2/0

  Service-policy output: POL-DIFFSERV
    ...

    Class-map: CLASS-DOS (match-all)
      313 packets, 27544 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 102
      police:
          cir 400000 bps, bc 12500 bytes
        conformed 313 packets, 27544 bytes; actions:
          set-dscp-transmit af12
        exceeded 0 packets, 0 bytes; actions:
          set-dscp-transmit default
        conformed 0 bps, exceed 0 bps
       ...

2002#show policy-map interface Serial 2/1
 Serial2/1

  Service-policy output: POL-DIFSERV-OUT
    ...

    Class-map: CLASS-DOS-DS (match-all)
      313 packets, 27544 bytes
      30 second offered rate 0 bps, drop rate 0 bps
      Match:  dscp af12 (12)
      Queueing
        Output Queue: Conversation 265
        Bandwidth 30 (%)
        Bandwidth 463 (kbps)Max Threshold 64 (packets)
        (pkts matched/bytes matched) 6/528
        (depth/total drops/no-buffer drops) 0/0/0
        ...
```

Luego se intenta generar con la herramienta `hping3(8)` mucho tráfico para la clase EF y poder ver que el exceso se descarta.

```
root@source:~# hping3 --fast --syn 4.0.0.1
HPING 4.0.0.1 (tap0 4.0.0.1): S set, 40 headers + 0 data bytes
len=40 ip=4.0.0.1 ttl=252 id=30264 sport=0 flags=RA seq=0 win=0 rtt=20.1 ms
...
root@source:~# hping3 --fast --syn 4.0.0.1 -p 23
HPING 4.0.0.1 (tap0 4.0.0.1): S set, 40 headers + 0 data bytes
len=44 ip=4.0.0.1 ttl=252 id=20902 sport=23 flags=SA seq=0 win=4128
    rtt=13.7 ms
len=44 ip=4.0.0.1 ttl=252 id=20855 sport=23 flags=SA seq=1 win=4128
    rtt=21.1 ms
...
^C
--- 4.0.0.1 hping statistic ---
152 packets transmitted, 151 packets received, 1% packet loss
round-trip min/avg/max = 13.1/18.6/54.9 ms
```

Se ve en el router que se producen los descartes.

```
2001#show policy-map interface Serial 2/0
 Serial2/0

  Service-policy output: POL-DIFFSERV

    Class-map: CLASS-UNO (match-all)
      359 packets, 15907 bytes
      5 minute offered rate 1000 bps, drop rate 0 bps
      Match: access-group 101
      police:
          cir 20000 bps, bc 1500 bytes
        conformed 359 packets, 15907 bytes; actions:
          set-dscp-transmit ef
        exceeded 0 packets, 0 bytes; actions:
        drop
        conformed 2000 bps, exceed 0 bps

    Class-map: CLASS-DOS (match-all)
      313 packets, 27544 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: access-group 102
      police:
          cir 400000 bps, bc 12500 bytes
        conformed 313 packets, 27544 bytes; actions:
      ...
```

Para las clase DOS, si se genera tráfico en exceso en lugar de descartarlo se configuró para que se lo remarque como “best-effort”. Con la herramienta ping se marcan con el valor 0x10 para ver que luego el exceso se remarca a 0x00.

```
2001(config)#interface Serial 2/0
2001(config-if)#load-interval 30

root@source:~# ping -i 0.0000001 -Q 0x10 -s 1400 &
               ping -i 0.0000001 -Q 0x10 -s 1400 &
               ping -i 0.0000001 -Q 0x10 -s 1400 &
...
721 packets transmitted, 718 received, 0% packet loss, time 10715ms
rtt min/avg/max/mdev = 14.112/24.643/233.745/28.306 ms, pipe 20,
ipg/ewma 14.883/24.296 ms

2001#show policy-map  interface Serial 2/0
 Serial2/0

  Service-policy output: POL-DIFFSERV

    Class-map: CLASS-UNO (match-all)
```

```
...

Class-map: CLASS-DOS (match-all)
  8100 packets, 11178528 bytes
  30 second offered rate 266000 bps, drop rate 0 bps
  Match: access-group 102
  police:
      cir 400000 bps, bc 12500 bytes
    conformed 2804 packets, 3594656 bytes; actions:
      set-dscp-transmit af12
    exceeded 5296 packets, 7583872 bytes; actions:
      set-dscp-transmit default
    conformed 52000 bps, exceed 213000 bps

Class-map: CLASS-TRES (match-all)
...
```

D.2. Ejemplo DiffServ sobre GNU/Linux IPv4

En esta sección se presentan ejemplos sobre la plataforma GNU/Linux usando la misma herramienta vista con los ejemplos de colas, TC. En los ejemplos del anexo sobre técnicas de encolado se mostró que para la herramienta de TC de GNU/Linux se asocia un parámetro llamado priomap (Priority Map). Este parámetro está directamente asociado a los valores de los bits de ToS que llevan los paquetes IP. El priomap es una secuencia de 16 números a partir de los cuales se deriva a que cola van a parar los paquetes sin necesidad de definir un filtro extra. Funciona a modo de filtro de acuerdo a la marca de ToS. Su definición se presta a menudo a confusión. Se tiende a pensar que el orden de los valores numéricos está asociado al orden de los valores de ToS. Si se piensa de esta manera se puede derivar de forma incorrecta la asociación. Para la salida default.

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

Se podría pensar en asociar: ToS 0x00 al la cola en la posición 1, ToS 0x02 con la cola en la pos. 2, y así hasta llegar al ToS 0x1E en la pos. 16. Por ejemplo, para el priomap default se creería que se asocia de la siguiente manera.

```
Orden:              01 02 03 04 05 06 07 08 09  10              16
valores de ToS:     00 02 04 06 05 06 08 0A 10 12  ...          1E
salida del comando: 1, 2, 2, 2, 1, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1
```

Los datagramas con la marca 0x00 van a la cola 1(Med), con 0x02 van a la cola 2(Low), los con 0x08 van a la cola 0(High).
Pero **NO** es así. Esta es una interpretación incorrecta. Lo que indica la salida del comando, como priomap no es el orden de las marca de ToS sino el orden de la prioridad Linux en orden creciente. Entonces la interpretación debería ser: los que

tienen prioridad 0 van a la cola 1, los de prioridad 1,2,3 van a la cola 2, los de prioridad 4 van a la cola 1, los de prioridad 5 a la cola 2, los de prioridad 6 y 7 a la 0, etc. Pero como por default no se usan todas las prioridades quedan asignaciones de prioridad a cola que no se utilizan. Por ejemplo, la prioridad 3 o la prioridad 5 no se usan. Por default solo se usan 5 prioridades: 0,1,2,4,6, las indicadas en el columna Linux Priority de la tabla D.1, donde también está la asociación de la prioridad con el ToS.

Valor ToS	Servicio	Linux Priority	Band/Cola
0x00	Normal Service	0 Best Effort	1
0x02	Minimize Monetary Cost	1 Filler	2
0x04	Maximize Reliability	0 Best Effort	1
0x06	mmc+mr	0 Best Effort	1
0x08	Maximize Throughput	2 Bulk	2
0x0a	mmc+mt	2 Bulk	2
0x0c	mr+mt	2 Bulk	2
0x0e	mmc+mr+mt	2 Bulk	2
0x10	Minimize Delay	6 Interactive	0
0x12	mmc+md	6 Interactive	0
0x14	mr+md	6 Interactive	0
0x16	mmc+mr+md	6 Interactive	0
0x18	mt+md	4 Int. Bulk	1
0x1a	mmc+mt+md	4 Int. Bulk	1
0x1c	mr+mt+md	4 Int. Bulk	1
0x1e	mmc+mr+mt+md	4 Int. Bulk	1

Cuadro D.1: Valores del Prioridades dentro de Linux de acuerdo al ToS

El Priomap se debe interpretar entonces como el siguiente texto:

```
Prioridad:              00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
salida del comando:      1, 2, 2, 2, 1, 2, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1
                         +  +  +     +     +
Prioridades Usadas:      0  1  2     4     6
```

Dando entonces prioridad 0 (Best Effort) van a la cola 1, los de 1,2 (Filler, Bulk) a la cola 2, los de prioridad 4 (Interactive Bulk) a la cola 1 y los de prioridad 6 (Interactive) van a la cola más prioritaria, la 0. Esto es lo que dice la documentación de GNU/Linux para el TC: 1, 2, 1, 1, 2, 2, 2, 2, 0, 0, 0, 0, 1, 1, 1, 1. Algunas referencias se pueden encontrar en las siguientes URLs:

- `http://linux.derkeiler.com/Newsgroups/comp.os.linux.networking/2004-06/0278.html`
- `http://osdir.com/ml/linux.network.routing/2002-04/msg00031.html`

Los ejemplos a continuación se muestran usando el comando `iptables(8)` que se puede utilizar para marcar el tráfico cuando sale del origen. En el caso del texto se utilizará más adelante pero en lugar de marcar en el origen se marcará en el borde del dominio DS.

```
## 0x10/0x3f telnet
iptables -t mangle -A OUTPUT -p tcp --dport 23    -j TOS
         --set-tos Minimize-Delay
iptables -t mangle -A OUTPUT -p tcp --dport 23    -j TOS
         --set-tos 0x10/0x3f
## 0x08/0x3f iperf
iptables -t mangle -A OUTPUT -p tcp --dport 5001  -j TOS
         --set-tos Maximize-Throughput
## 0x04/0x3f
iptables -t mangle -A OUTPUT -p tcp --dport 5002  -j TOS
         --set-tos Maximize-Reliability
## 0x02/0x3f NNTP
iptables -t mangle -A OUTPUT -p tcp --dport 119   -j TOS
         --set-tos Minimize-Cost
## 0x00/0x3f ICMP
iptables -t mangle -A OUTPUT -p icmp -j TOS
         --set-tos Normal-Service
```

Para ver en funcionamiento el clasificador basado en el priomap se debe cambiar la qdisc default `pfifo_fast` por una prio, simplemente para poder ver el matching del tráfico en las diferentes colas. En realidad, el sistema `pfifo_fast` ofrece similar funcionalidad que un prio, pero no permite ver las sub-colas o clases. A continuación se puede ver que con FIFO no se ven las clases, sub-colas o bandas.

```
root@n2# tc qdisc show
qdisc pfifo_fast 0: dev eth0 root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc pfifo_fast 0: dev eth1 root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

root@n2# tc -s class show dev eth1
```

Agregar la qdisc prio e inspeccionar.

```
root@n2# tc qdisc add dev eth1 root handle 1: prio bands 3

root@n2# tc qdisc show
qdisc pfifo_fast 0: dev eth0 root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
qdisc prio 1: dev eth1 root refcnt 2 bands 3
      priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1

root@n2# tc class show dev eth1
class prio 1:1 parent 1:
class prio 1:2 parent 1:
class prio 1:3 parent 1:
```

Best-effort va a parar a la cola 1 (uno).

```
root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 528 bytes 6 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Agregar el marcado en el router **2001** o nodo **n1**, en el ejemplo con GNU/Linux. En este caso, en lugar de usar `OUTPUT`, se usa `FORWARD` ya que se marca cuando llega al borde del dominio de QoS y no en el origen.

```
root@n1#
iptables -t mangle -A FORWARD -p tcp --dport 23    -j TOS
         --set-tos Minimize-Delay
iptables -t mangle -A FORWARD -p tcp --dport 5001  -j TOS
         --set-tos Maximize-Throughput

root@n1# iptables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source
           destination
    0     0 TOS        tcp  --  *      *       0.0.0.0/0
            0.0.0.0/0            tcp dpt:23 TOS set 0x10/0x3f
    0     0 TOS        tcp  --  *      *       0.0.0.0/0
            0.0.0.0/0            tcp dpt:5001 TOS set 0x08/0x3f
```

Siguiendo con el ejemplo se genera el tráfico desde el origen.

```
root@n5# nc 4.0.0.1 23
HOLA
QUE TAL
^C
```

Ver el matching en el dominio de QoS DS.

```
root@n1# iptables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 10 packets, 549 bytes)
 pkts bytes target     prot opt in     out     source
      destination
  6   325 TOS        tcp  --  *      *       0.0.0.0/0
      0.0.0.0/0            tcp dpt:23 TOS set 0x10/0x3f
    0     0 TOS        tcp  --  *      *       0.0.0.0/0
      0.0.0.0/0            tcp dpt:5001 TOS set 0x08/0x3f

root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 409 bytes 6 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 2670 bytes 31 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Generar el tráfico para la clase 2 (dos).

```
root@n5# iperf -c 4.0.0.1
------------------------------------------------------------
Client connecting to 4.0.0.1, TCP port 5001
TCP window size: 21.0 KByte (default)
------------------------------------------------------------
[  3] local 20.0.0.100 port 42075 connected with 4.0.0.1 port 5001
^C[ ID] Interval       Transfer     Bandwidth
[  3]  0.0- 0.7 sec  7.00 MBytes  88.4 Mbits/sec
```

Ver el matching del tráfico.

```
root@n1# iptables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 7865 packets, 7781K bytes)
 pkts bytes target     prot opt in     out     source
      destination
    6   325 TOS        tcp  --  *      *       0.0.0.0/0
      0.0.0.0/0            tcp dpt:23 TOS set 0x10/0x3f
```

```
5094 7614K TOS        tcp  --  *      *       0.0.0.0/0
      0.0.0.0/0            tcp dpt:5001 TOS set 0x08/0x3f

root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 409 bytes 6 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 3944 bytes 46 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 7684956 bytes 5094 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

D.3. Ejemplo DiffServ con ToS sobre GNU/Linux IPv6

Para IPv6 con la herramienta TC, los filters no funcionan de forma automática de acuerdo a las marcas ToS y el priomap. Para que funcione hay que hacerlo de forma manual. Según la documentación pareciera que deberían ir a dar a bandas separadas, el tráfico con diferente ToS, pero no funciona así. El problema parece ser que el modelo no se aplica a IPv6 en GNU/Linux. Según la documentación encontramos: **How come that IPv6 tc filters do not work?** pudiéndose consultar en la URL `http://lartc.org/lartc.html#AEN1446`. Traduciendo al español: "...La base de datos de políticas de ruteo, Routing Policy Database (RPDB), en IPv6 reemplaza la estructura de direcciones y ruteo del Kernel lo que da varias nuevas funcionalidades. Desafortunadamente la estructura para IPv6 en Linux es implementada fuera del kernel, por lo que la RPDB no participa en la mayor parte del manejo de los paquetes IPv6, por eso el funcionamiento es distinto..." En conclusión los priomap **NO** funcionan de forma adecuada en GNU/Linux para IPv6. Para verlo, si configura para marca tráfico con ToS en el borde.

```
root@n1#
ip6tables -t mangle -A FORWARD -p tcp --dport 23    -j TOS
         --set-tos Minimize-Delay
ip6tables -t mangle -A FORWARD -p tcp --dport 5001  -j TOS
         --set-tos Maximize-Throughput

root@n1# ip6tables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source
      destination
    0     0 TOS        tcp      *      *       ::/0
      ::/0                tcp dpt:23 TOS set 0x10/0x3f
    0     0 TOS        tcp      *      *       ::/0
      ::/0                tcp dpt:5001 TOS set 0x08/0x3f
```

Los contadores originales son lo mostrados.

```
root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
Sent 528 bytes 6 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 17398 bytes 199 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 7684956 bytes 5094 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Luego se genera el tráfico.

```
root@n5# nc -6 2001:db8:4::1 23

# ip6tables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 14 packets, 1040 bytes)
 pkts bytes target     prot opt in     out     source
      destination
    8   600 TOS        tcp      *      *       ::/0
    ::/0                 tcp dpt:23 TOS set 0x10/0x3f
    0     0 TOS        tcp      *      *       ::/0
    ::/0                 tcp dpt:5001 TOS set 0x08/0x3f

root@n5# telnet 2001:db8:4::1 5001
Trying 2001:db8:4::1...

root@n1# ip6tables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 26 packets, 1880 bytes)
 pkts bytes target     prot opt in     out     source
      destination
    8   600 TOS        tcp      *      *       ::/0
      ::/0                 tcp dpt:23 TOS set 0x10/0x3f
    6   480 TOS        tcp      *      *       ::/0
      ::/0                 tcp dpt:5001 TOS set 0x08/0x3f
```

Se ve que no cambia la banda sobre la que se coloca el tráfico. Quedan con los valores originales.

```
root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
Sent 528 bytes 6 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 17398 bytes 199 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 7684956 bytes 5094 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Se encontró que bajo algunas aplicaciones, con marcado desde origen, por ejemplo, telnet parece funcionar, pero no es para todos los casos. Para que funcione con las marcas de `ip6tables` se deben coloca filtros explícitos en lugar de usar el priomap. En este caso no se está considerando que se puede marcar tráfico con ECN. Sino se debería cambiar la máscara por 0xFC.

```
tc filter add dev eth1 parent 1: protocol ipv6 prio 1 u32 match
          ip6 priority 0x10 0xff flowid 1:1
tc filter add dev eth1 parent 1: protocol ipv6 prio 2 u32 match
          ip6 priority 0x00 0xff flowid 1:2
tc filter add dev eth1 parent 1: protocol ipv6 prio 3 u32 match
          ip6 priority 0x08 0xff flowid 1:3
```

Luego volver a generar tráfico y ver el matching.

```
root@n5# nc -6 2001:db8:4::1 23
HOLA
222
^C

root@n1# ip6tables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 114 packets, 8085 bytes)
 pkts bytes target     prot opt in     out     source
   destination
   25  1929 TOS        tcp      *      *       ::/0
   ::/0                 tcp dpt:23 TOS set 0x10/0x3f
   13  1040 TOS        tcp      *      *       ::/0
   ::/0                 tcp dpt:5001 TOS set 0x08/0x3f
```

Ahora los contadores sí se incrementan.

```
root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 4514 bytes 50 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 31700 bytes 362 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 7684956 bytes 5094 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

Ver generar tráfico para la otra clase.

```
root@n5# nc -6 2001:db8:4::1 5001
SSSS
SSS
^C
```

También se incrementan los contadores.

```
root@n1# ip6tables -L -n -v -t mangle | grep -A4 FORWARD
Chain FORWARD (policy ACCEPT 124 packets, 8830 bytes)
 pkts bytes target     prot opt in     out     source
            destination
   25  1929 TOS        tcp      *      *       ::/0
            ::/0                 tcp dpt:23 TOS set 0x10/0x3f
   19  1489 TOS        tcp      *      *       ::/0
            ::/0                 tcp dpt:5001 TOS set 0x08/0x3f

root@n2# tc -s class show dev eth1
class prio 1:1 parent 1:
 Sent 4514 bytes 50 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:2 parent 1:
 Sent 32490 bytes 371 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
class prio 1:3 parent 1:
 Sent 7685489 bytes 5100 pkt (dropped 0, overlimits 0 requeues 0)
 backlog 0b 0p requeues 0
```

D.4. Ejemplo DiffServ con DSCP sobre GNU/Linux IPv6

El ejemplo a continuación en lugar de usar la codificación de ToS se realiza con el DSCP. Se replica el mismo ejemplo que se realizó sobre la plataforma cisco, pero en este caso con IPv6. La política a implementar en los nodos de borde es la siguiente.

```
policy-map POL-DIFFSERV
 class CLASS-UNO
    police 20000 conform-action set-dscp-transmit ef
    exceed-action drop
 class CLASS-DOS
    police 400000 conform-action set-dscp-transmit af12
                  exceed-action set-dscp-transmit default
 class CLASS-TRES
    police 800000 conform-action set-dscp-transmit af22
                  exceed-action set-dscp-transmit default
 class class-default
  fair-queue
```

Los valores a marcar para cada tipo de tráfico son los siguientes.

```
ef    = 0x2e  TOS=0xb8     DSCP 46/0x2e  MASK=0xfc
af12  = 0x0c  TOS=0x30     DSCP 12/0x0c  MASK=0xfc
af22  = 0x14  TOS=0x50     DSCP 20/0x14  MASK=0xfc
```

Para el marcado se puede utilizar la misma herramienta, `ip6tables`.

```
root@n1#
ip6tables -A FORWARD -t mangle -p tcp --dport 23 -d  2001:DB8:4::1/128
          -j DSCP --set-dscp 0x2e
ip6tables -A FORWARD -t mangle -p ipv6-icmp      -d  2001:DB8:4::100/128
          -j DSCP --set-dscp 0x0c
ip6tables -A FORWARD -t mangle -p tcp --dport 23 -d  2001:DB8:4::50/128
          -j DSCP --set-dscp 0x14
```

Para el policing se debe tener en cuenta que en los tests se coloca la cola default con ID 4000 y no con 40 como debería ser. De esta forma no se hacen los matching para el tráfico distinto del especificado explícitamente. Las reglas serían las siguientes.

```
root@n1#
 tc qdisc add dev eth1 root handle 1: htb default 4000
 tc class add dev eth1 parent 1: classid 1:1 htb rate 10mbit

 tc class add dev eth1 parent 1:1 classid 1:10 htb rate 20000bit
    ceil 20000bit burst 10k prio 1
 tc class add dev eth1 parent 1:1 classid 1:20 htb rate 400000bit
    ceil   10mbit burst 10k prio 2
 tc class add dev eth1 parent 1:1 classid 1:30 htb rate 800000kbit
    ceil   10mbit burst 10k prio 2
 tc class add dev eth1 parent 1:1 classid 1:40 htb rate 10mbit
    ceil   10mbit burst 10k prio 3
```

Para calcular los parámetros para RED.

```
root@n1# bc
bw=10000
dl=200
avpkt=1500
min=bw*dl/8
max=min*2
limit=4*max
burst=(min*2+max)/(3*avpkt)
min
250000
max
500000
limit
2000000
avpkt
1500
burst
222
quit

root@n1#
 tc qdisc add dev eth1 parent 1:40 handle 4000: red limit 2000000
    min 250000 max 500000 avpkt 1500 burst 222 bandwidth 10mbit ecn
```

Ver el estado.

```
root@n1# tc class show dev eth1
class htb 1:1 root rate 10000Kbit ceil 10000Kbit
      burst 1600b cburst 1600b
class htb 1:10 parent 1:1 prio 1 rate 20000bit ceil 20000bit
      burst 10Kb cburst 1600b
class htb 1:20 parent 1:1 prio 2 rate 400000bit ceil 10000Kbit
      burst 10Kb cburst 1600b
class htb 1:30 parent 1:1 prio 2 rate 800000Kbit ceil 10000Kbit
```

```
      burst 10100b cburst 1600b
class htb 1:40 parent 1:1 leaf 4000: prio 3 rate 10000Kbit ceil 10000Kbit
      burst 10Kb cburst 1600b
class red 4000:1 parent 4000:

root@n1# tc qdisc show dev eth1
qdisc htb 1: root refcnt 2 r2q 10 default 40 direct_packets_stat 832
qdisc red 4000: parent 1:40 limit 2000000b min 250000b max 500000b ecn
```

Luego se debe indicar las acciones del overlimit o el exceso. Estas pueden ser.

continue: causa que no se aplique nada, pero puede seguir probando con otras reglas que están por debajo.

drop: se descarta el tráfico en exceso.

Pass/OK: pasa el tráfico sin seguir probando con otras reglas.

reclassify: se re-escribe, re-clasifica como "best-effort". Es la acción default.

Entonces la configuración en el borde sería algo como lo siguiente.

```
## CLASE-UNO
tc filter add dev eth1 parent 1: protocol ipv6 prio 1 u32 match
   ip6 dst 2001:DB8:4::1/128
   match ip6 protocol 6 0xff match ip6 dport 23 0xffff flowid 1:10
   police rate 20000bps burst 10240 action drop
## CLASE-DOS
tc filter add dev eth1 parent 1: protocol ipv6 prio 2 u32
   ip6 dst 2001:DB8:4::100/128
   match ip6 protocol 58 0xff flowid 1:20
   police rate 400000bps burst 10240 action reclassify
## CLASE-TRES
tc filter add dev eth1 parent 1: protocol ipv6 prio 3 u32 match
   ip6 dst 2001:DB8:4::50/128
   match ip6 protocol 6 0xff match ip6 dport 23 0xffff flowid 1:30
   police rate 800000bps burst 10240 action reclassify
```

Luego restaría implementar la política en los nodos de core.

```
 policy-map POL-DIFSERV-OUT
  class CLASS-UNO-DS
   priority percent 20
  class CLASS-DOS-DS
   bandwidth percent 30
  class CLASS-TRES-DS
   bandwidth percent 10
  class class-default
   fair-queue
  random-detect
```

Para el policing, tener en cuenta que para no usar la default se coloca el valor 4000 en lugar del 40. En la sintaxis de GNU/Linux podría ser.

```
root@n2#
 tc qdisc add dev eth1 root handle 1: htb default 4000
 tc class add dev eth1 parent 1: classid 1:1 htb rate 10mbit
 tc class add dev eth1 parent 1:1 classid 1:10 htb rate 2mbit
    ceil  2mbit burst 10k prio 1
 tc class add dev eth1 parent 1:1 classid 1:20 htb rate 3mbit
    ceil 10mbit burst 10k prio 2
 tc class add dev eth1 parent 1:1 classid 1:30 htb rate 1mbit
```

```
    ceil 10mbit burst 10k prio 2
 tc class add dev eth1 parent 1:1 classid 1:40 htb rate 10mbit
    ceil 10mbit burst 10k prio 3
 tc qdisc add dev eth1 parent 1:40 handle 4000: red limit 2000000 min 250000
    max 500000 avpkt 1500 burst 222 bandwidth 10mbit ecn
```

Y el filtrado de acuerdo a las marcas, en GNU/Linux se hace con el valor del ToS dado a partir de los DSCP: ef=0x2e/0xfc, af12=0x0c/0xfc y af22=0x14/0xfc, dando entonces los valores por byte completo: ef=0xb8/0xfc, af12=0x30/0xfc y af22=0x50/0xfc. La configuración sería la siguiente.

```
## CLASE-UNO
tc filter add dev eth1 parent 1: protocol ipv6 prio 1 u32 match ip6
   priority 0xb8 0xfc flowid 1:10
## CLASE-DOS
tc filter add dev eth1 parent 1: protocol ipv6 prio 2 u32 match ip6
   priority 0x30 0xfc flowid 1:20
## CLASE-TRES
tc filter add dev eth1 parent 1: protocol ipv6 prio 3 u32 match ip6
   priority 0x50 0xfc flowid 1:30

root@n2# tc class show dev eth1
class htb 1:1 root rate 10000Kbit ceil 10000Kbit burst 1600b cburst 1600b
class htb 1:10 parent 1:1 prio 1 rate 2000Kbit ceil 2000Kbit
      burst 10Kb cburst 1600b
class htb 1:20 parent 1:1 prio 2 rate 3000Kbit ceil 10000Kbit
      burst 10Kb cburst 1600b
class htb 1:30 parent 1:1 prio 2 rate 1000Kbit ceil 10000Kbit
      burst 10Kb cburst 1600b
class htb 1:40 parent 1:1 leaf 4000: prio 3 rate 10000Kbit ceil 10000Kbit
      burst 10Kb cburst 1600b
class red 4000:1 parent 4000:

root@n2# tc qdisc show dev eth1
qdisc htb 1: root refcnt 2 r2q 10 default 40 direct_packets_stat 479
qdisc red 4000: parent 1:40 limit 2000000b min 250000b max 500000b ecn
```

Luego se pueden realizar las pruebas y ver como se encola en las diferentes bandas. Antes de realizar las pruebas se ve que los contadores están en cero y, debido a que la default se envió a la 4000, no se ven ni los paquetes de control, ruteo.

```
root@n1# ip6tables -L -t mangle -v | grep DSCP
    0     0 DSCP       tcp          any    any     anywhere
    2001:db8:4::1/128    tcp dpt:telnet DSCP set 0x2e
    0     0 DSCP       ipv6-icmp    any    any     anywhere
    2001:db8:4::100/128  DSCP set 0x0c
    0     0 DSCP       tcp          any    any     anywhere
    2001:db8:4::50/128   tcp dpt:telnet DSCP set 0x14

root@n1# tc -s class show dev eth1 | grep Sent
 Sent 1936 bytes 22 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 1936 bytes 22 pkt (dropped 0, overlimits 0 requeues 0)
```

Luego generar los flujos. Por ejemplo, para la primera clase.

```
root@n5# telnet 2001:db8:4::1
Trying 2001:db8:4::1...

root@n1# ip6tables -L -t mangle -v | grep DSCP
4   320 DSCP       tcp          any    any     anywhere
        2001:db8:4::1/128    tcp dpt:telnet DSCP set 0x2e
    0     0 DSCP       ipv6-icmp    any    any     anywhere
        2001:db8:4::100/128  DSCP set 0x0c
    0     0 DSCP       tcp          any    any     anywhere
        2001:db8:4::50/128   tcp dpt:telnet DSCP set 0x14

root@n2# tc -s class show dev eth1 | grep Sent
 Sent 376 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 376 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
```

Para la segunda clase.

```
root@n5# ping6 2001:db8:4::100 -c 10
PING 2001:db8:4::100(2001:db8:4::100) 56 data bytes
64 bytes from 2001:db8:4::100: icmp_seq=1 ttl=61 time=1.24 ms
64 bytes from 2001:db8:4::100: icmp_seq=2 ttl=61 time=0.568 ms
64 bytes from 2001:db8:4::100: icmp_seq=3 ttl=61 time=0.842 ms
64 bytes from 2001:db8:4::100: icmp_seq=4 ttl=61 time=0.602 ms
64 bytes from 2001:db8:4::100: icmp_seq=5 ttl=61 time=0.551 ms
64 bytes from 2001:db8:4::100: icmp_seq=6 ttl=61 time=0.567 ms
64 bytes from 2001:db8:4::100: icmp_seq=7 ttl=61 time=0.623 ms
64 bytes from 2001:db8:4::100: icmp_seq=8 ttl=61 time=0.600 ms
64 bytes from 2001:db8:4::100: icmp_seq=9 ttl=61 time=0.661 ms
64 bytes from 2001:db8:4::100: icmp_seq=10 ttl=61 time=0.587 ms

--- 2001:db8:4::100 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/mdev = 0.551/0.684/1.240/0.201 ms

root@n1# ip6tables -L -t mangle -v | grep DSCP
    4   320 DSCP       tcp          any    any     anywhere
        2001:db8:4::1/128    tcp dpt:telnet DSCP set 0x2e
10  1040 DSCP       ipv6-icmp    any    any     anywhere
       2001:db8:4::100/128  DSCP set 0x0c
    0     0 DSCP       tcp          any    any     anywhere
       2001:db8:4::50/128   tcp dpt:telnet DSCP set 0x14

root@n2 tc -s class show dev eth1 | grep Sent
 Sent 1556 bytes 14 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 376 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
Sent 1180 bytes 10 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
```

Para la tercera.

```
root@n5# telnet 2001:db8:4::50
Trying 2001:db8:4::50...

root@n1# ip6tables -L -t mangle -v | grep DSCP
    4   320 DSCP       tcp          any    any     anywhere
        2001:db8:4::1/128    tcp dpt:telnet DSCP set 0x2e
   10  1040 DSCP       ipv6-icmp    any    any     anywhere
       2001:db8:4::100/128  DSCP set 0x0c
4   320 DSCP       tcp          any    any     anywhere
        2001:db8:4::50/128   tcp dpt:telnet DSCP set 0x14

root@n2# tc -s class show dev eth1 | grep Sent
 Sent 1932 bytes 18 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 376 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
```

```
 Sent 1180 bytes 10 pkt (dropped 0, overlimits 0 requeues 0)
Sent 376 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
```

Y en el equipo de borde se ven todos los matching.

```
root@n1:/tmp/pycore.46183/n1.conf# tc -s class show dev eth1 | grep Sent
 Sent 1932 bytes 18 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 376 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 1180 bytes 10 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 376 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
```

Para ver los matching en la default, re-cargar todas las reglas, pero cambiando solo en la primera del 4000 por el 40.

```
root@n1#
 tc qdisc del dev eth1 root handle 1: htb default 4000
 tc qdisc add dev eth1 root handle 1: htb default 40
 ...

root@n2#
 tc qdisc del dev eth1 root handle 1: htb default 4000
 tc qdisc add dev eth1 root handle 1: htb default 40
 ...
```

Ahora se puede ver el matching de la default.

```
root@n2# tc -s class show dev eth1 | grep Sent
 Sent 23892 bytes 258 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 1534 bytes 13 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 0 bytes 0 pkt (dropped 0, overlimits 0 requeues 0)
 Sent 22358 bytes 245 pkt (dropped 0, overlimits 0 requeues 0)
```

Con este se finalizan los ejemplos de los apéndices.

Bibliografía

[AC11] S. Amante and B. Carpenter. Rfc-6438: Using the ipv6 flow label for equal cost multipath routing and link aggregation in tunnels, 2011.

[AJC11] S. Amante, S. Jiang, and B. Carpenter. Rfc-6436: Rationale for update to the ipv6 flow label specification, 2011.

[AJCR11] S. Amante, S. Jiang, B. Carpenter, and J. Rajahalme. Rfc-6437: Ipv6 flow label specification, 2011.

[Alm92] P. Almquist. Rfc-1349: Type of service in the internet protocol suite, 1992.

[AR09] L. Andersson and Asati R. Rfc-5462: Multiprotocol label switching (mpls) label stack entry: .^{ex}p"field renamed to "traffic class"field, 2009.

[BBC+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. Rfc-2475: An architecture for differentiated service, 1998.

[BCS94] R. Braden, D. Clark, and S. Shenker. Rfc-1633: Integrated services in the internet architecture: an overview, 1994.

[Ber99] S. Berson. Rsvp and integrated services with ipv6 flow labels. ietf internet draft. draft-berson-rsvp-ipv6-fl-00, 1999.

[BILFD01] F. Baker, C. Iturralde, F. Le Faucheur, and B. Davie. Rfc-3175: Aggregation of rsvp for ipv4 and ipv6 reservations, 2001.

[Bla09] S. Blake. Use of the ipv6 flow label as a transport-layer nonce to defend against off-path spoofing attacks. ietf internet draft. draft-blake-ipv6-flow-label-nonce-02, 2009.

[BMFM96] Matthias C.F. Birkner, Liam Murphy, Matthias C. F, and Birkner Liam Murphy. Impres: Supporting services in ipv6 using the flow label and hop-by-hop option fields, 1996.

[Bol98] G. Bolch. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. Wiley-Interscience Publication. 1998.

[BP95] L.S. Brakmo and L.L. Peterson. Tcp vegas: End to end congestion avoidance on a global internet. pages 1465–1480, Washington, DC, USA, 1995. IEEE Computer Society.

[BS02a] Rahul Banerjee and Sethuraman. Design and implementation of the quality-of-service in ipv6 using the modified hop-by-hop extension header - a practicable mechanism. ietf internet draft. draft-banerjee-ipv6-quality-service-02, 2002.

[BS02b] Rahul Banerjee and Sethuraman. A radical approach in providing quality-of-service over the internet using the 20-bit ipv6 flow label field. ietf internet draft. draft-jagadeesan-rad-approach-service-01, 2002.

[BSM02] Rahul Banerjee, Paul Malhotra Sumeshwar, and M. Mahaveer. A modified specification for use of the ipv6 flow label for providing an efficient quality of service using a hybrid approach. ietf internet draft. draft-banerjee-flowlabel-ipv6-qos-03, 2002.

[BZ96] J. Bennet and H. Zhang. Wf2q: Worst-case fair weighted fair queueing. pages 120–182, Washington, DC, USA, 1996. IEEE Computer Society.

[BZB+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Rfc-2205: Resource reservation protocol (rsvp) – version 1 functional specification, 1997.

[CBB08] S. Chakravorty, J. Bush, and Bond. Ipv6 label switching architecture. ietf internet draft. draft-chakravorty-6lsa-03, 2008.

[CDG06] A. Conta, S. Deering, and M. Gupta. Rfc-4443: Internet control message protocol (icmpv6) for the internet protocol version 6 (ipv6) specification, 2006.

[CJT14] B. Carpenter, S. Jiang, and W. Tarreau. Rfc-7098: Using the ipv6 flow label for load balancing in server farms, 2014.

[CL03] Hyojeong Choe and S.H. Low. Stabilized vegas. In *Proceedings of IEEE INFOCOM 2005*, volume 3, pages 2290–2300 vol.3, March 2003.

[CR01a] A. Conta and J. Rajahalme. A model for diffserv use of the ipv6 flow label specification. ietf internet draft. draft-conta-diffserv-ipv6-fl-classifier-01, 2001.

[CR01b] A. Conta and J. Rajahalme. A proposal for the ipv6 flow label specification. ietf internet draft. draft-conta-ipv6-flow-label-02, 2001.

[Cra98] M. Crawford. Rfc-2464: Transmission of ipv6 packets over ethernet networks, 1998.

[DCB+02] B. Davie, A. Charny, J. Bennett, K. Benson, J. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. Rfc-3246: An expedited forwarding phb (per-hop behavior), 2002.

[DE11] C. Donley and K. Erichsen. Using the flow label with dual-stack lite. ietf internet draft. draft-donley-softwire-dslite-flowlabel-02, 2011.

[DHJ+04] S. Deering, B Haberman, T. Jinmei, E Nordmark, and B. Zill. Rfc-4007: Ipv6 scoped address architecture, 2004.

[FDP+05] El-Bahlul Fgee, Kenney Jason D., William J. Phillips, William Robertson, and S. Sivakumar. Comparison of qos performance between ipv6 qos management model and intserv and diffserv qos models, 2005.

[FGS01] S. Floyd, R. Gummadi, and S. Shenker. Adaptive red: an algorithm for increasing the robustness of red's active queue management. preprint available at http://www.icir.org/floyd, 2001.

[FJ93] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. Washington, DC, USA, 1993. IEEE Computer Society.

[FJ95] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. pages 365–386, Washington, DC, USA, 1995. IEEE Computer Society.

[FKR+04] El-Bahlul Fgee, Jason D. Kenney, W. Robertson, W. J. Phillips, and S. Sivakumar. Implementing an ipv6 qos management scheme using flow label & class of service fields. *Computer Science and Information Engineering, World Congress on*, 2:1049–1052 Vol.2, 2004.

[FPRS03] E.-B. Fgee, W. J. Phillips, W. Robertson, and S. C. Sivakumar. Implementing qos capabilities in ipv6 networks and comparison with mpls and rsvp. In *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*, volume 2, pages 851–854 vol.2, May 2003.

[FS03] S. Fahmy and Mridul Sharma. Deployment considerations for the tcp vegas congestion control algorithm. http://docs.lib.purdue.edu/cstech/1571, 2003.

[FV03] W. Feng and S. Vanichpun. Enabling compatibility between tcp reno and tcp vegas. In *IEEE Proceeding of Symposium on Applications and the Internet (SAINT03)*, pages 301–308, 2003.

[Gro02] D. Grossman. Rfc-3260: New terminology and clarifications for diffserv, 2002.

[GTB95] R. Gilligan, S. Thomson, and J. Bound. Ipv6 program interfaces for bsd systems draft-ietf-ipngwg-bsd-api-00, 1995.

[GTB+03] R. Gilligan, S. Thomson, J. Bound, J. McCann, and W. Stevens. Rfc-3493: Basic socket interface extensions for ipv6, 2003.

[GTBS97] R. Gilligan, S. Thomson, J. Bound, and W. Stevens. Rfc-2133: Basic socket interface extensions for ipv6, 1997.

[GTBS99] R. Gilligan, S. Thomson, J. Bound, and W. Stevens. Rfc-2553: Basic socket interface extensions for ipv6, 1999.

[HA98] D. Haskin and E. Allen. Rfc-2472: Ip version 6 over ppp, 1998.

[Hab02] B. Haberman. Rfc-3307: Allocation guidelines for ipv6 multicast addresses, 2002.

[Hag01] J. Hagino. Socket api for ipv6 flow label field. ietf internet draft. draft-itojun-ipv6-flowlabel-api-01, 2001.

[Hag06] Silvia Hagen. *IPv6 Essentials.* O'Reilly Media, Inc., 2006.

[Has89] E. Hashem. Analysis of random drop for gateway congestion control, report lcs tr-465, laboratory for computer science, mit, cambridge, ma, 1989, p.103, 1989.

[HB11] Q. Hu and Carpenter B. Rfc-6294: Survey of proposed use cases for the ipv6 flow label, 2011.

[HBG00] U. Hengartner, J. Bolliger, and T. Gross. Tcp vegas revisited. In *Proceedings of IEEE INFOCOM '00*, pages 1546–1555 Vol. 3, 2000.

[HBWW99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Rfc-2597: Assured forwarding phb group, 1999.

[HC04] C. Hitema and B. Carpenter. Rfc-3879: Deprecating site local addresses, 2004.

[HD98] R. Hinder and S. Deering. Rfc-2460: Internet protocol, version 6 specification., 1998.

[HD06] R. Hinden and S. Deering. Rfc-4291: Ip version 6 addressing architecture, 2006.

[HH05] R Hinden and B. Haberman. Rfc-4193: Unique local ipv6 unicast addresses, 2005.

[iJi04] Hangino itojun Jun-ichiro. *IPv6 Network Programming.* Elsevier Digital Press, 2004.

[Jac] V. Jacobson. Modified tcp congestion avoidance algorithm. technical report.

[JNP99] V. Jacobson, K. Nichols, and K. Poduri. Rfc-2598: An expedited forwarding phb, 1999.

[JPBM10] J. Jeong, S. Park, L. Beloeil, and S. Madanapalli. Rfc-6106: Pv6 router advertisement options for dns configuration, 2010.

[KB00] Ibrahim Khalil and Torsten Braun. Implementation of a bandwidth broker for dynamic end-to-end resource reservation in outsourced virtual private networks. In *in Outsourced Virtual Private Networks. The 25th Annual IEEE Conference on Local Computer Networks (LCN*, pages 9–10, 2000.

[KR12] J. Kurose and K. Ross. *Computer Networking: A Top-Down Approach, 6th. Ed.* Addison-Wesley, 2012.

[LK04] I. Lee and S. Kim. A qos improvement scheme for real-time traffic using ipv6 flow labels. volume 3043, pages 278–285. Springer, Berlin Heidelberg, 2004.

[Los03] Pete Loshin. *IPv6, 2nd Edition*. Morgan Kaufmann, Burlington, MA, USA, 2003.

[LS98] M.V. Loukola and J.O. Slytta. New possibilities offered by ipv6. In *Computer Communications and Networks, 1998. Proceedings. 7th International Conference on*, pages 548–552, Washington, DC, USA, Oct 1998. IEEE Computer Society.

[LTH06] C. Lin, P. Tseng, and W. Hwang. End-to-end qos provisioning by flow label in ipv6. Atlantis Press, 2006.

[MBDL99] May Martin, Bolot Jean Bolot, C. Diot, and Lyles. 'reasons not to deploy red. reasons not to deploy red. In *Proc. of 7th. International Workshop on Quality of Service (IWQoS'99)*, pages 548–552, 1999.

[MBM13] Luis Marrone, Andres Barbieri, and Robles Matías. Tcp performance - cubic, vegas & reno. In *JCS&T, Vol.13, No.1*, pages pp:1–8, 50 y 120, La Plata, Argentina, April 2013. Fac. de Informática, UNLP.

[MF05] J. Manner and X. Fu. Rfc-4094: Analysis of existing quality-of-service signaling protocols, 2005.

[MH00] J. Metzler and S. Hauth. An end-to-end usage of the ipv6 flow label. ietf internet draft. draft-metzler-ipv6-flowlabel-00, 2000.

[MLAW99] Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand. Analysis and comparison of tcp reno and vegas. In *Proceedings of IEEE INFOCOM '99*, pages 1556–1563, 1999.

[MM05] O. McGann and D. Malone. Flow label filtering feasibility. In *EC3ND 2005 Proceedings of the First European Conference on Computer Network Defence School of Computing, University of Glamorgan, Wales, UK*, pages 41–49. Springer London, June 2005.

[MN06] Ian Mcdonald and Richard Nelson. Congestion control advancements in Linux. In *Linux Conf AU*, 2006.

[MSSH11] S. McFarland, M. Sambi, N Sharma, and S. Hooda. *IPv6 for Enterprise Networks, 1st. Ed.* Cisco Press, 2011.

[NBBB98] K. Nichols, S. Blake, F. Baker, and D. Black. Rfc-2474: Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers, 1998.

[NDK07] T. Narten, R. Draves, and S. Krishnan. Rfc-4941: Privacy extensions for stateless address autoconfiguration in ipv6, 2007.

[NJZ99] K. Nichols, V. Jacobson, and L. Zhang. Rfc-2638: A two-bit differentiated services architecture for the internet, 1999.

[NNSS07] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Rfc-4861: Neighbor discovery for ip version 6 (ipv6), 2007.

[PE05] Eric C. K. Poh and Hong Tat Ewe. Ipv6 packet classification based on flow label, source and destination addresses. In *ICITA '05: Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume 2*, pages 659–664, Washington, DC, USA, 2005. IEEE Computer Society.

[Pfe07] René Pfeiffer. Tcp and linux'pluggable congestion control algorithms. *Linux Gazette*, (135), Feb 2007.

[PHPH05] J. Padilla, Monica Huerta, Josep Paradells, and Xavier Hesselbach. Intserv6: An approach to support qos over ipv6 networks. In *ISCC '05: Proceedings of the 10th IEEE Symposium on Computers and Communications*, pages 77–82, Washington, DC, USA, June 2005. IEEE Computer Society.

[PLAG06] Ciprian Popoviciu, Eric Levy-Abegnoli, and Patrick Grossetete. *Deploying IPv6 Networks.* Cisco Press, 2006.

[Pra04] Bahanu Prakash. Using the 20 bit flow label field in the ipv6 header to indicate desirable quality of service on the internet, 2004.

[RC01] J. Rajahalme and Conta. An ipv6 flow label specification proposal. ietf internet draft. draft-rajahalme-ipv6-flow-label-00, 2001.

[RCCD04] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering. Rfc-3697: Ipv6 flow label specification (obsoleted), 2004.

[RFB01] K. Ramakrishnan, S. Floyd, and D. Black. Rfc-3168: The addition of explicit congestion notification (ecn) to ip, 2001.

[Rob08] Matías Robles. Qos en redes wireless con ipv6, 2008.

[RVC01] E. Rosen, A. Viswanathan, and R. Callon. Rfc-3031: Multiprotocol label switching architecture, 2001.

[SB03] Günther Stattenberger and Torsten Braun. Performance of a bandwidth broker for diffserv networks. In *Kommunikation in verteilten Systemen (KiVS03*, pages 25–28, 2003.

[Sch94] Mischa Schwartz. *Redes de telecomunicaciones, protocolos, modelado y análisis*. Addison-Wesley Iberoamericana, 1994.

[SFR03] W. Richard Stevens, B. Fenner, and A. Rudoff. *The Sockets Networking API: UNIX Network Programming Volume 1, 3rd Edition*. Addison-Wesley, 2003.

[SHM08] Qiong Sun, Xiaohong Huang, and Yan Ma. Ipv6 end-to-end qos provision in heterogeneous networks using flow label. In *MMIT '08: Proceedings of the 2008 International Conference on MultiMedia and Information Technology*, pages 621–624, Washington, DC, USA, 2008. IEEE Computer Society.

[SHM09] Qiong Sun, Xiaohong Huang, and Yan Ma. Ipv6 end-to-end qos provision in heterogeneous networks using aggregated flow label. *Computer Science and Information Engineering, World Congress on*, 2:438–441, 2009.

[SJA05] K. N. Srijith, Lillykutty Jacob, and A. L. Ananda. Tcp vegas-a: Improving the performance of tcp vegas. *ACM Computer Communications*, 28(4):429–440, mar 2005.

[SK02] Pasi Sarolahti and Alexey Kuznetsov. Congestion control in linux TCP. In *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference*, pages 49–62, Berkeley, CA, USA, 2002. USENIX Association.

[SPG97] S. Shenker, C. Partridge, and R. Guerin. Rfc-2212 - specification of guaranteed quality of service, 1997.

[SS05] J. Sing and B. Soh. Tcp new vegas: Improving the performance of tcp vegas over high latency links. In *Network Computing and Applications, 4th IEEE International Symposium on*, pages 73–82, July 2005.

[Sto06] Benedikt Stockebrand. *IPv6 in Practice (A Unixer's Guide to the Next Generation Internet*. Springer-Verlag Berlin Heidelberg, 2006.

[SW97] S. Shenker and J. Wroclawski. Rfc-2215 rfc-2215 - general characterization parameters for integrated service network elements, 1997.

[TNJ07] S. Thomson, T. Narten, and T. Jinemi. Rfc-4862: Ipv6 stateless address autoconfiguration, 2007.

[TOLa] Ns-2: Network simulator 2 http://www.isi.edu/nsnam/ns/.

[TOLb] Opnet: http://www.opnet.com/.

[TOLc] Qossim: Quality of service simulator.

[Wro97a] L. Wroclawski. Rfc-2210 - the use of rsvp with ietf integrated services, 1997.

[Wro97b] L. Wroclawski. Rfc-2211 - specification of the controlled-load network element, 1997.

[WZDW09] Haixing Wang, Ke Zhang, Weifeng Du, and Liufang Wang. Implementation of ipv6 network qos based on intserv and snmp. In *Proceedings of the 2009 International Symposium on Information Processing (ISIP 09)*, pages 117–12, Huangshan, P. R. China, Agust 2009. ACADEMY PUBLISHER.

[ZC01] Wang Zheng and David Ed. Clark. *Internet QoS Architectures and Mechanisms for Quality of Service*. The Morgan Kaufmann Series in Networking. Ed David Clark, Burlington, MA, USA, 2001.

[Zha90] L. Zhang. Virtual clock: A new traffic control algorithm for packet swithing networks. pages 19–29, Philadephia, PA, 1990. ACM.

[ZXWZ12] Wei Zhou, Wei Xing, Yongchao Wang, and Jianwei Zhang. Tcp vegas-v: Improving the performance of tcp vegas. In *Automatic Control and Artificial Intelligence (ACAI 2012), International Conference*, pages 2034–2039, Washington, DC, USA, March 2012. IEEE Computer Society.