Implementación de IPv6, QoS e IPSec con Linux

Cristina Durán Sanlés

Manuel David Fernández Vaamonde

27 de mayo de 2003

Índice

1. Introducción **4**

2. IPv6 **6**

2.1. Configuración del kernel y utilidades IPv6 6

2.2. Situación inicial de los interfaces de red 7

2.3. Configuración de una dirección IPv6 8

2.4. Configuración de la tabla de rutas 9

2.5. Túneles sit para IPv6 11

2.5.1. Configuración de túneles sit 12

2.6. Algunas curiosidades alrededor de IPv6 13

2.6.1. IPv6 y servicios de red 13

2.6.2. Usando IPv6 en internet 14

3. QoS **16**

3.1. QoS en Linux 16

3.2. Configuración del kernel para QoS 19

3.3. Introducción al marcado de paquetes con *iptables* 21

3.4. La herramienta *tc* 22

3.4.1. Escenario de ejemplo 23

3.4.2. Comentarios finales acerca de QoS en Linux 29

4. IPsec **30**

4.1. La importancia de la encriptación: SSH, IPsec y demás hierbas 30

4.2. Aplicaciones de IPsec 31

4.3. Instalando FreeSwan en el kernel de Linux 32

4.4. Configurando IPsec/FreeSwan 34

4.5. Algo más sobre IPsec/FreeSwan 39

5. Conclusiones finales **41**

6. Referencias bibliográficas **43**

6.1. IPv6 43

6.2. QoS 43

6.3. IPsec 43

Índice de figuras

1. Esquema de la técnica de “encolado” 17

2. Estructura del árbol de clases 25

3. Estructura del árbol de clases con préstamo de tráfico 27

1. Introducción

Este trabajo, presentado para la asignatura *Redes*, impartida en la Facultad de Informática de La Coruña, pretende presentar de manera totalmente práctica y basándose en un operativo, con el cual cualquier persona podría hacer pruebas sin ningún coste, un recorrido por la implementación de diversas partes del temario impartido en la asignatura.

Trataremos tres de los grandes temas tratados en la segunda parte del curso como son **IPv6**, **QoS** e **IPsec** y veremos su implantación en un sistema GNU/Linux (en particular con una distribución Debian, aunque este hecho será totalmente irrelevante para lo explicado).

En particular en la parte de **IPv6**, explicaremos las opciones a activar en un kernel de Linux para dar soporte a esta pila de protocolos, como activar un interfaz, configurar un túnel IPv6-over-IPv4 (sit) o añadir una ruta

En la parte de **QoS**, explicaremos como realizar control de tráfico de salida en nuestra red, de manera que podamos (usando métodos de marcado en firewall) controlar nuestra red para dar prioridades a cierto tipo de tráfico.

En la parte de **IPsec**, explicaremos la implantación de un enlace IPsec entre dos ordenadores, quizás una de las formas más básicas de montaje de este servicio, pero que ilustra a la perfección la filosofía de la misma. La implantación de IPsec se realizará con FreeSwan, una implementación libre para Linux.

Para realizar los ejemplos se usará la herramienta general de configuración de redes *iproute* que nos proveerá de un método avanzado de configuración de las mismas, salvando algunas carencias que podemos encontrar en las herramientas clásicas *ifconfig* y *route*.

De este modo, a través del comando *ip*, podremos realizar la configuración de las interfaces de red, de los túneles y del rutado con una herramienta única, potente y de manera muy homogénea.

2. IPv6

2.1. Configuración del kernel y utilidades IPv6

Para dar soporte de IPv6 a una máquina Linux, hemos de empezar por activar en el kernel ciertas opciones que nos permitirán que nuestra máquina soporte esta pila de protocolos.

Es interesante comentar que no es necesario compilar estas características *built-in* en el kernel, sino que se pueden compilar como módulos para añadirlas posteriormente a nuestro antojo.

La opción a activar para dar soporte bien como modulo, bien *built-in* en el kernel de Linux está localizada en la sección *Networking Options* y aparece de la siguiente manera:

```
<M>   The IPv6 protocol (EXPERIMENTAL)
```

Una vez hayamos compilado el kernel con este soporte o bien añadido el módulo (en nuestro caso, por usar Debian, hacemos uso de la herramienta *modconf*), tendremos habilitado el soporte IPv6 en nuestra máquina Linux.

Llegados a este punto es interesante hacer notar que hemos de adaptar nuestras herramientas de red a esta nueva situación, por ejemplo, el ping clásico no nos permite realizar pings a direcciones de tipo IPv6. Para ello, en nuestro caso hemos tenido que instalar paquetes adicionales con estas nuevas utilidades, por ejemplo *iputils-ping* que nos proporciona *ping6* herramienta para poder hacer pings a direcciones IPv6 o *iputils-tracepath*, que nos proporciona *tracepath6* una evolución de la herramienta *traceroute* para uso con direcciones IPv6.

Una vez instaladas estas utilidades, podemos comenzar a hacer pruebas con IPv6 y comprobar que nuestro sistema se ha configurado correctamente.

2.2. Situación inicial de los interfaces de red

Una vez hemos dado soporte en el kernel, podemos observar el estado de las direcciones de los interfaces del siguiente modo:

```
$ ip addr show
```

Suponiendo que tengamos una ethernet con interfaz *eth0*, veremos la siguiente configuracion, sin haber operado sobre ella ningún cambio:

```
eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
  link/ether 00:e0:7d:d0:5c:c7 brd ff:ff:ff:ff:ff:ff
  inet 192.168.2.2/24 brd 192.168.1.255 scope global eth0
  inet6 fe80::2e0:7dff:fed0:5cc7/10 scope link
```

Podemos ver que por defecto tenemos dos direcciones sobre este interfaz, la dirección IPv4 y la dirección IPv6 de enlace que se añade a los interfaces cuando tenemos soporte de este stack de protocolos.

También podemos observar que estar dirección está compuesta por los tres últimos bytes de la dirección mac, que se puede observar en la línea que comienza por *link/ether*.

Vemos también el ámbito de la dirección, junto al campo *scope*, que en este caso es de *link*, pero podría ser también *global* o de *site*.

Es interesante comentar que con el comando *ip* se diferencia entre una dirección (*addr*) y un enlace (*link*), de los cuales podríamos extraer información con la siguiente orden:

```
$ ip link show
```

2.3. Configuración de una dirección IPv6

Para configurar un interfaz con una dirección IPv6 usaremos la herramienta *ip* de nuevo, aunque se podría realizar con *ifconfig*. Para ello usamos el comando *add*, sobre el objeto *addr*, veamos un ejemplo:

```
$ ip addr add 2001:618:4:2000::10f4/127 dev eth0
```

Hemos de incluir la dirección y el prefijo de la misma, que se asociará a nuestra interfaz de red. Una de las novedades en el uso de IPv6 es que los interfaces de red siempre tendrán varias direcciones de red, ya que como mínimo siempre existirá la dirección de enlace.

Si una vez añadida la dirección encuestamos al interfaz sobre sus direcciones como hemos visto anteriormente, obtenemos el siguiente resultad:

```
eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
 link/ether 00:e0:7d:d0:5c:c7 brd ff:ff:ff:ff:ff:ff
 inet 192.168.2.2/24 brd 192.168.1.255 scope global eth0
```

```
inet6 fe80::2e0:7dff:fed0:5cc7/10 scope link
inet6 2001:618:4:2000::10f4/127 scope global
```

donde podemos observar que se ha añadido una dirección *global* de red, de tipo IPv6.

Podemos ahora realizar un ping al interfaz para comprobar su correcto funcionamiento del siguiente modo:

```
$ ping6 2001:618:4:2000::10f4
```

Obteniendo el resultado esperado si la configuración ha sido correcta.

2.4. Configuración de la tabla de rutas

Como en los ejemplos anteriores seguiremos usando el comando `ip`, sin embargo, también es posible usar el antiguo comando `route` con el parámetro `-A inet6` para introducir la ruta.

Para ver las rutas IPv6 actuales con el comando `ip`, usaremos la siguiente orden:

```
$ ip -6 route show
```

que nos mostrará las rutas en la actual máquina.

Para añadir una ruta IPv6 usaremos el comando `add`, sobre el objeto `route`, un ejemplo:

```
$ ip route add ::/0 via 2001:618:4:2000::10f4
```

Este es el ejemplo mas simple que podemos poner, recordemos que el rutado en IPv6 se realiza mediante prefijos que van haciendo "match" en la tabla de rutas, en este caso, esta ruta producirá el enrutamiento de todo el tráfico a través de esa dirección de red. Podríamos si quisiéremos complicar el ejemplo tanto como deseemos para adaptar nuestra máquina a la estructura de la organización.

Una vez añadida la ruta, podemos usar la orden vista anteriormente, obteniendo el siguiente resultad:

```
2001:618:4:2000::10f4/127 dev eth0  proto kernel  metric 256  mtu 1500
fe80::/10 dev eth0  proto kernel  metric 256  mtu 1500
ff00::/8 dev eth0  proto kernel  metric 256  mtu 1500
default dev eth0  proto kernel  metric 256  mtu 1500
default via 2001:618:4:2000::10f4 dev eth0  metric 1024  mtu 1500
```

Es interesante observar las rutas `fe80` que es la ruta al interfaz de link y `ff00` como ruta multicast.

De esta manera podríamos llegar ya a las máquinas en nuestro segmento de red y sería muy sencillo (y así se ha hecho en las pruebas para este trabajo

) configurar dos máquinas con par trenzado cruzado para que se comuniquen entre si usando IPv6.

2.5. Túneles sit para IPv6

En la implantación de IPv6 es habitual el uso de túneles, generalmente por motivos de compatibilidad, hemos de pensar que IPv6 no está muy extendido y en muchos casos es habitual que haya que usar transporte IPv4 para llegar a zonas de uso IPv6, este es el tipo de túnel que mostraremos en este caso.

La mecánica es la habitual en todos los túneles, rellenaremos el campo de datos de un paquete IPv4 con paquetes IPv6 que una vez hayan llegado al otro punto del transporte, se extraerán y dirigirán correctamente a las máquinas IPv6.

La herramienta `ip` nos permite hacer tres tipos de túneles:

- Túneles `ipip`, donde se encapsula IPv4 sobre IPv4 para determinadas aplicaciones.
- Túneles `gre`, donde se realiza un tráfico encriptado sobre IP. (Menos potente que IPsec).
- Túneles `sit`, donde se encapsula IPv6 sobre IPv4 de modo que podamos establecer una comunicación de un punto a otro punto con IPv4 pero que transportará IPv6.

En nuestro caso usaremos túneles `sit`, para realizar comunicación IPv6 entre dos redes, teniendo como transporte IPv4.

Por otra parte, estamos hablando de un tunnel que establece una conexión punto a punto entre dos máquinas de modo que hemos de levantar el tunnel en ambas máquinas para que este sea completo.

2.5.1. Configuración de túneles sit

Para la configuración de un tunnel IPv6-over-IPv4 usaremos como en los ejemplos anteriores la herramienta `ip`, con esta, para ver los túneles que existen en la máquina, podemos usar los siguientes comandos:

```
ip -6 tunnel show
```

Para levantar un túnel IPv6 sobre IPv4, hemos de operar sobre el objeto `tunnel`, de la siguiente forma:

```
$ ip tunnel add sit1 mode sit ttl 12 remote 192.168.2.2 local 192.168.2.1
```

en una de las máquinas, y en la otra:

```
$ ip tunnel add sit1 mode sit ttl 12 remote 192.168.2.1 local 192.168.2.2
```

De este modo hemos configurado el túnel en ambos lados de la máquina, dándole de nombre `sit1` en ambas (podría ser cualquiera), con tipo de

tunnel (`mode`), `sit`, e indicando cual será nuestra dirección IPv4 en ambos extremos del tunnel.

Ahora solo debemos levantar ambos lados del tunnel de la siguiente manera:

```
$ ip link set dev sit1 up
```

Una vez hecho esto, solo nos quedará redirigir nuestro tráfico IPv6 hacia este tunnel del siguiente modo:

```
$ ip -6 route add ::/0 dev sit1 metric1
```

Con esto conseguiremos que nuestro tráfico se enrute a través del túnel y sea transportado a través de IPv4

2.6. Algunas curiosidades alrededor de IPv6

2.6.1. IPv6 y servicios de red

Como curiosidad de esta experiencia de montar una máquina Linux con IPv6 hemos sacado algunas notas "heurísticas" que es preciso tener en cuenta.

En primer lugar saber que muchos de los programas que usarán la red en Linux vienen preparados para trabajar con IPv6, esto es todo un alivio

para alguien que pretende usarla, pero también hemos de tener en cuenta que generalmente estos vienen "de serie" configurados para IPv4 y es habitual tener que realizar algún cambio.

Un ejemplo: Para acceder a una máquina IPv6 con `ssh` hemos de añadir la opción `-6` del siguiente modo:

```
$ ssh -6 davidfv@2001:618:4:2000::10f4
```

Sin embargo, para usar ssh también el servidor (`sshd`) ha de estar preparado. Por defecto ssh esta escuchando solamente en una dirección ip de llegada (ha hecho "bind" en una ip concreta), que podría ser por ejemplo `0.0.0.0`, de modo que lo haría en todas, para hacer que nos permita llegar con una dirección IPv6 hemos de modificar el fichero de configuración `sshd_config` con la siguiente línea:

```
ListenAddress ::
```

Y así escuchará en cualquier dirección IPv6 que tenga la máquina del servidor.

2.6.2. Usando IPv6 en internet

Como curiosidad introducimos la posibilidad de que, hoy en día y de manera gratuita, ya es posible usar IPv6 para acceder a sitios de internet

que tenga esta posibilidad. Existen ciertos servidores denominados "brokers" que, previo registro (gratuito), te dan la posibilidad de establecer un túnel IPv6 sobre IPv4 desde tu máquina hasta la suya, de modo que puedas rutar tu tráfico IPv6 a través de ellos que lo sacarán al backbone IPv6.

Como muestra hemos usado el servicio dado por `http://www.freenet6.net`. En esta página nos podemos bajar un cliente para Linux que se encarga de autenticarnos ante el servidor de túneles de freenet6 para que podamos realizar el túnel contra el.

Realmente, el propio cliente se encarga de realizar el tunnel y añadir la ruta correcta, pero es una buena oportunidad de ver el funcionamiento de uno de estos servicios, pudiendo acceder a través de los túneles a servicios como por ejemplo el irc de Openprojects (`irc.ipv6.freenode.net`).

3. QoS

3.1. QoS en Linux

Linux, desde hace ya bastantes versiones, nos permite desarrollar distintos tipo de QoS para controlar el tráfico saliente y entrante en nuestra red. La diferenciación de servicio como tal existe en Linux aunque no implementada completamente, pero es muy común entre los administradores de sistemas el uso de herramientas de calidad de servicio (*traffic control*) desde la propia máquina, de forma que se pueda asegurar que cierto tipo de tráfico de nuestra red, que saldrá por una máquina Linux haciendo las veces de router tendrá prioridades que los demás tipos. En resumen, controlar los flujos de red que se produzcan, desde nuestra red y hacia nuestra red.

El control de tráfico de esta manera con máquinas Linux, se está haciendo extremadamente popular en muchas organizaciones debido a los problemas que pueden suponer por una parte los programas *p2p* que están muy de moda y que pueden colapsar el tráfico de un red (si alguien ha probado alguno de estos programas en su casa, se dará cuenta que rápidamente satura la salida a red por ejemplo de un navegador) y por otra parte porque ciertos programas pueden ser de una importancia crucial o necesitar una latencia baja, frente a otros programas (ejemplo de herramientas de administración como PCAnyWhere o ssh, en el que se desea que las respuestas sean lo más rápidas posibles para facilitar la sesión.)

Para controlar el tráfico saliente de nuestra red se utiliza la técnica de **"encolado"** (*queueing.*

El tráfico se marca de algún modo (generalmente cuando pasa a través

del firewall) y posteriormente se introduce, según estas marcas, en la clase ("disciplina de cola") que tenga asociado ese identificador, asociada a un interfaz de red.

En las clases se estructura el tráfico con un comportamiento determinado, como veremos posteriormente y cada clase tiene una cola asociada con un comportamiento también parametrizable y determinado que saca el tráfico a la red.

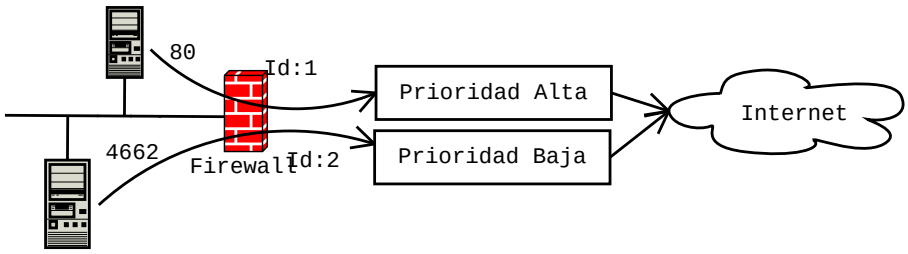

Figura 1: Esquema de la técnica de "encolado"

Como tipo de colas según su algoritmo, podemos mencionar:

- **pfifo_fast**: Clásica cola, "primero en entrar, primero en salir", esta clase tiene tres bandas (0, 1, 2 y 3), de modo que mientras exista un paquete en la banda 0, ninguna de los paquetes en la 1, 2 o 3 podrá ser trasmitido.

  Esta cola mira el tipo de servicio (TOS) del paquete que llega y sitúa los paquetes marcados como "mínimo retardo" en la banda 0, esto es lo único que la diferencia de una cola común.

- **Stochastic Fairness Queuing**: Es una implementación del algoritmo de "encolado justo" de modo que trata de repartir el ancho de banda equitativamente entre todos los demandantes del servicio.

Parar realizar esto, crea muchas subcolas FIFO a las que va permitiendo mandar progresivamente con un mecanismo similar al de *Round Robin*.

.

Si hablamos de comportamientos de clases:

- **Hierarchical Token Bucket**: Este algoritmo nos permite dividir el ancho de banda especificando ancho de banda máximo y mínimo.

  El algoritmo nos asegura el ancho de banda mínimo y si este no está siendo usado garantiza el ancho de banda máximo.

  Esta disciplina es especialmente útil para no solo garantizar el uso de un determinado ancho de banda, sino que cuando el tráfico no está siendo usado, pueda ser utilizado por las demás clases.

- **CBQ**: Este algoritmo (especialmente complejo) fue usado durante mucho tiempo como algoritmo de clase (disciplina de cola) cuando HTB no era incluida en el kernel.

  Básicamente el algoritmo trata de parar el interfaz para conseguir el ratio fijado en la cola.

  Se aconseja, como veremos en los ejemplos siguientes, el uso de HTB en lugar de CBQ por su sencillez y por proveer las mismas facilidades.

Estas no son las únicas disciplinas que hay, existen muchas otras, pero son suficientes para la gran mayoría de las situaciones que se nos puedan ocurrir y para ilustrar el modo de implantar control de tráfico.

3.2. Configuración del kernel para QoS

En primer lugar hemos de tener en el kernel las opciones que permiten el uso de *iptables* (netfilter) y su módulo de marcado de paquetes de modo que habremos de marcar las siguientes opciones en la sección *Network Options*

```
[*] Network packet filtering (replaces ipchains)

IPv6: Netfilter Configuration  --->
  <*>   netfilter MARK match support
  <*>   Packet filtering
  <*>   Packet mangling
     <*>     MARK target support
```

Una vez que hemos activado el soporte de iptables y su modulo de marcado de paquetes (*MARK*), hemos de activar el soporte para QoS del kernel. Es interesante hacer notar que el soporte de algunas modalidades de cola como la HTB no han aparecido en el kernel hasta la versión 2.4.20, de modo que las pruebas para este trabajo están siendo efectuadas con esta versión.

En la sección *Network Options*, hemos de activar las siguientes opciones:

```
QoS and/or fair queueing  --->
 [*] QoS and/or fair queueing
 [*] SFQ queue
 [*] HTB queue
 <*>   Ingress Qdisc
```

```
[*]   QoS support
[*]     Rate estimator
[*]   Packet classifier API
 <*>     TC index classifier
 <*>     Routing table based classifier
 <*>     Firewall based classifier
```

Con estas opciones (bien como modulo las que no los permita, bien en el propio kernel), activadas, ya estaremos en disposición de comenzar la configuración de calidad de servicio en nuestro equipo.

Si en un futuro necesitásemos más tipos de colas, hemos de habilitar en este último grupo de opciones las que deseemos.

3.3. Introducción al marcado de paquetes con *iptables*

Una vez que tenemos soporte para el kernel, lo primero que debemos es aprender a marcar el tráfico de nuestra red, de modo que quede totalmente caracterizado y podamos decidir posteriormente a que cola deseamos enviarlo para que se produzca su salida a la red.

La herramienta en la que nos vamos a basar para el marcado de tráfico es *iptables*. *iptables* es una evolución de la herramienta de *ipchains*, que gestiona el firewall de Linux, el NAT del sistema y también puede tratar paquetes para realizar algún cambio (en el TOS o, como es nuestro caso añadiéndoles una marca).

Igual que *ipchains* se basa en reglas con unos determinados atributos de modo que si un paquete de red coincide con los atributos que caracterizan una determinada regla, el paquete se tratará conforme al comportamiento indicado en esa regla.

Veamos un ejemplo de marcado de paquetes, sobre el que realizaremos las explicaciones[1]:

```
$ iptables -A POSTROUTING -o eth0 -p tcp --dport 80 \
           -t mangle -j MARK --set-mark 1
```

Con esta regla estamos diciendo que el tráfico que ya haya pasado por las tablas de enrutamiento (POSTROUTING), que salga por el interfaz eth0 (

[1]Nótese que en la bibliografía, en el artículo de Bulma, existen varias incorreciones que ya han sido notificadas a su autor.

-o), con protocolo tcp (-p), y que acceda a una página web (-dport 80), se use la tabla de manejo de paquetes (-t mangle), se cargue el modulo de marcado (-j MARK) y se marque con el número 1 (-set-mark).

Si por ejemplo tenemos una situación clásica por la cual, varias máquinas están saliendo a través de la nuestra (una máquina Linux actuando de router o haciendo NAT), hemos de realizar el marcado en la cadena FORWARD en lugar de en POSTROUTING ya que el paquete solo lo estamos reenviando, no "accionamos" sobre el.

Un ejemplo de regla de este tipo:

```
$ iptables -A FORWARD -s 192.168.0.1 -t mangle -j MARK --set-mark 1
```

Todo lo que reenviemos, que llegue desde la ip 192.168.0.1 (-s), lo marcamos con 1.

De este modo podemos caracterizar nuestro propio tráfico o el de máquinas de las que nos encargaremos de reenviar sus paquetes.

3.4. La herramienta *tc*

La herramienta *tc* nos servirá para controlar, las *clases*, las *colas* y los *filtros*. Estos tres elementos nos ayudarán a componer nuestro escenario de control de tráfico. Recordemos: los **filtros** los colocaremos para indicar que paquetes con una determinada marca "caen" a una clase.

En cada **clase** indicaremos un comportamiento de limitación de tráfico o

una determinada disciplina de clase y posteriormente asociaremos a esa clase un tipo de *cola* que mandará los paquetes a la red.

De modo que, una vez hemos visto como marcar paquetes de red, solo nos queda para completar nuestro escenario las piezas comentadas anteriormente: como estructurar un árbol de clases, las colas a asociar a cada una de esas clases y los filtros que decidirán que paquetes con que marcas se envían a las clases.

3.4.1. Escenario de ejemplo

Para explicar el funcionamiento de la herramienta *tc*, nos vamos a basar en un escenario bastante común en el que podemos aplicar calidad de servicio. El escenario es una máquina Linux que saca el trafico de otras dos (el número de máquinas será irrelevante en este escenario), nos interesa dar prioridad sobre el demás tráfico a ssh y a web.

Las máquinas conectadas al router, lo hacen por sus interfaces *eth1* (192.168.1.2) y *eth2* (192.168.1.3), el interfaz que da salida a internet lo hace a través del *eth0* (192.168.1.1).

En primer lugar, hemos de marcar el tráfico que nos interesa caracterizar, en este caso el destinado al puerto 80 y al puerto 22 de cualquier máquina. Lo podemos hacer del siguiente modo:

```
iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 80 \
-tmangle -j MARK --set-mark 1

iptables -A FORWARD -i eth2 -o eth0 -p tcp --dport 80 \
```

```
-t mangle -j MARK --set-mark 1

iptables -A FORWARD -i eth1 -o eth0 -p tcp --dport 22 \
-t mangle -j MARK --set-mark 1

iptables -A FORWARD -i eth2 -o eth0 -p tcp --dport 22 \
-t mangle -j MARK --set-mark 1
```

Marcamos con un 1 todos los paquetes que nos lleguen por los interfaces eth1 y eth2 y que salgan a través de eth0, destinados al puerto 80 o al 22 de alguna máquina.

En segundo lugar hemos de construir las **clases**. Las clases se estructuran de forma jerárquica (en forma de árbol), con una clase padre y distintas clases "hoja" a las que iremos mandando nuestro tráfico dependiendo de la marca que hayamos elegido y que tendrán ciertas colas asociadas.

En el ejemplo que usaremos para presentar el uso de *tc* para implantar calidad de servicio necesitamos dos tipos de tráfico, uno prioritario que requiere suficiente ancho de banda para funcionar de modo fluido y otro menos prioritario al que no hemos de permitir ocupar más ancho de banda de un mínimo permitido (el prioritario será el tráfico ssh y web y el menos prioritario el resto).

Para esto asociaremos al interfaz un árbol con dos clases distintas a las que se enviará el tráfico dependiendo de sus marcas.

Para la creación del nodo raíz (creación de la disciplina de cola), usamos la herramienta *tc*, veamos un ejemplo:

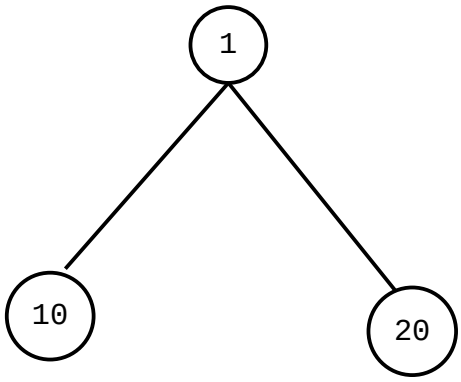

Figura 2: Estructura del árbol de clases

```
$ tc qdisc add dev eth0 root handle 1: default 20
```

Añadimos el nodo raíz de una disciplina de cola al interfaz eth0, lo nombramos como "1:" e indicamos que por defecto se envíe el tráfico a la clase 20.

La disciplina de cola es necesaria para colgar las clases propiamente dichas de ella y para indicar a que clase por defecto se enviará el tráfico.

Posteriormente hemos de colgar las clases necesarias de este raíz, en las que caracterizaremos el comportamiento del tráfico que recaiga en ellas.

```
tc class dev eth0 parent 1: classid 1:10 rate 150kbps ceil 300kbps
```

```
tc class dev eth0 parent 1: classid 1:20 rate 100kbps ceil 300kbps
```

Hemos creado dos clases a las cuales hemos puesto como nodo padre el anteriormente creado (`parent 1:`), hemos indicado su identificador de

clase y hemos fijado su ancho de banda garantizado (`rate`) y el máximo que pueden alcanzar (`ceil`).

Este ejemplo es sencillo y nos permite ver la filosofía del control de tráfico que es, enviar el tráfico marcado a distintas clases con unas características determinadas.

Ahora podríamos asociar *colas* a esas clases:

```
tc qdisc add dev eth0 parent 1:10 handle 10: sfq

tc qdisc add dev eth0 parent 1:20 handle 20: sfq
```

En este caso asociamos una "cola justa" a cada clase de modo que una vez que el tráfico ha ido a parar allí se reparta a la red a través de una cola con algoritmo *sfq*.

Finalmente hemos de configurar un *filtro*, que indique, llegado un paquete con una determinada marca, a que clase lo hemos de enviar, en nuestro caso:

```
tc filter add dev eth0 protocol ip parent 1: handle 1 fw classid 1:10
```

En la configuración de este filtro que asociamos a la clase 1: e interfaz eth0, indicamos que cualquier paquete con la marca (*handle*), 1, se reenvíe a la clase 10 (nuestra clase con más ancho de banda).

Este sería un ejemplo práctica de uso de calidad de servicio, de modo que garantizaremos prioridad de salida al trafico web y a ssh, frente a otros servicios de red.

Sin embargo, aunque hemos dicho que el algoritmo *htb* permite el uso de ancho de banda si este está libre, fijémonos en nuestro ejemplo: una de las clases tiene asegurado un ancho de banda de 100 kbps y la otra de 150kbps, de modo que el único tráfico que podría quedar libre son 50kbps ya que no la clase raíz puede prestar tráfico, ni una "hermana" puede pedir tráfico prestado de otra.

Con esta configuración, si una de las clases no estuviese recibiendo tráfico, estaríamos desperdiciando el ancho de banda de ella.

Para solucionar esto, podemos colocar un nodo intermedio entre el raíz y ambas clases que cope el total del tráfico y este si nos podrá "prestar" ancho de banda.

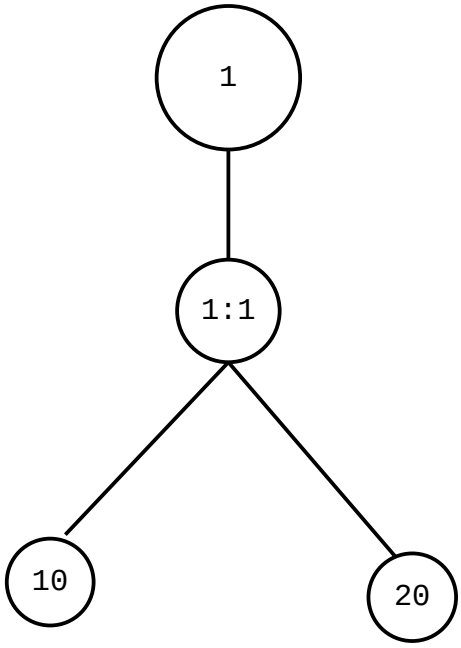

Figura 3: Estructura del árbol de clases con préstamo de tráfico

Para recrear esta situación, podríamos usar las siguientes órdenes:

```
tc qdisc add dev eth0 root handle 1: htb default 20

tc class add dev eth0 parent 1: classid 1:1 htb rate 300kbps ceil 300kbps

tc class add dev eth0 parent 1:1 classid 1:10 htb rate 150kbps ceil 300kbps

tc class add dev eth0 parent 1:1 classid 1:20 htb rate 100kbps ceil 300kbps

tc qdisc add dev eth0 parent 1:10 handle 10: sfq

tc qdisc add dev eth0 parent 1:20 handle 20: sfq

tc filter add dev eth0 protocol ip parent 1: handle 1 fw classid 1:10
```

Con las que conseguimos la situación mostrada. Para monitorizar el estado de las cosas, los paquetes que van "cayendo" en cada una de las clases, etcétera, podemos usar el comando *ls* del objeto *qdisc*, del siguiente modo:

```
tc -s qdisc ls dev eth0
```

De esta manera conseguiremos asegurar ancho de banda y aprovechar los anchos de banda no usados de otras clases.

3.4.2. Comentarios finales acerca de QoS en Linux

Lo comentado en este trabajo acerca de QoS es la punta del iceberg de lo que supone una implantación completa de calidad de servicio en Linux, si bien hemos expuesto el caso más clásico de uso de esta facilidad.

Por un lado comentar que Linux comienza a tener soporte para diferenciación de servicio tal como se ha estudiado en la asignatura y posiblemente comience a implantarse en algunos lugares en lugar del (simple) mecanismo que hemos puesto en esta sección.

Por otra parte no hemos abordado más algoritmos que existen, tanto de colas como de clases ya que nuestra pretensión al contar este tipo de QoS que realiza Linux es más enseñar las posibilidades comenzando desde un punto sencillo que explorar a fondo todo lo que nos ofrece la calidad de servicio en Linux.

Ya para finalizar, realizar una reflexión: hemos realizado todos nuestros ejemplos realizando control de tráfico sobre el tráfico de *salida* para conseguir una calidad de servicio razonable, pero para conseguir controlar realmente el tráfico y tener el dominio completo del reparto de anchos de banda de nuestra red es necesario también controlar el tráfico de entrada.

Lamentablemente, la cola *ingress* que es la usada para el control de tráfico entrante, no esta todavía bien documentada ni con los suficientes ejemplos para hablar de ella y conlleva una complejidad añadida que se escapa del propósito de este trabajo, si bien es necesario dejar clara la necesidad de tratar el tráfico de entrada para conseguir una calidad de servicio óptima.

4. IPsec

4.1. La importancia de la encriptación: SSH, IPsec y demás hierbas

Con la eclosión de internet y con el creciente auge de la misma para comunicación general, cobra un especial sentido la ocultación de nuestras comunicaciones.

Si bien han aparecido numerosas modificaciones de programas para que estos incorporen criptografía: servidores de correo con TLS o SASL, navegadores que soportan SSL o herramientas como SSH que permiten una sesión interactiva encriptada, la idea subyacente es la necesidad de implantar seguridad a nivel de red y no a nivel de aplicación.

Sin embargo IPv4 no contempla en su diseño la seguridad a nivel de red. Ni la encriptación de la información ni la autenticación (asegurar que ambos puntos de la comunicación son quienes dicen ser).

IPsec surge como un estándar de seguridad que añade estas dos funcionalidades a los protocolos de red, opcional en IPv4 y obligatorio en las implementaciones IPv6. Este estándar ha sido implementado por distintos productos permitiendo la interoperabilidad entre ellos, es muy conocida por ejemplo la implementación de CISCO, aunque otras compañías conocidas como Lucent o 3COM han hecho su propia implementación de IPsec. Nosotros en este trabajo implantaremos una red privada virtual con una implementación para Linux llamada **FreeSwan**.

El funcionamiento de IPsec se basa en **túneles encriptados y autenticados**. Por un lado se establece un túnel previa autenticación con clave

pública entre dos máquinas y posteriormente se envían los datos a través del túnel de modo que dentro del segmento de datos del paquete "normal" de red, incluimos un paquete de datos completo encriptado en origen y que será desencriptado en destino (y rutado si es necesario).

Mediante este mecanismo convertimos una red potencialmente insegura como internet, en una red segura sobre la que se pueden implantar ciertos servicios que hasta este momento serían impensables sobre una red de este tipo.

4.2. Aplicaciones de IPsec

Como hemos dicho, IPsec abre un abanico sumamente interesante de posibilidades, las dos principales son las *Redes Privadas Virtuales (VPN)* y los *"RoadWarriors"*.

Las *Redes Privadas Virtuales*, permiten comunicar varias redes locales entre si mediante el uso de un medio inseguro como es internet usando sus direcciones IP locales como si estuviesen dentro de nuestra red físicamente y siempre de un modo seguro.

Como *"RoadWarriors"* se denominan a aquellos puntos itinerantes de nuestra red (comerciales con portátiles, trabajadores que realizan su trabajo desde casa, conexiones desde hoteles, etc..) a los que le aseguramos que la comunicación entre el punto en el que se encuentre y su lugar de trabajo, a través de internet, se realizará de un modo seguro sin ser posible que sea interceptada provocando el descubrimiento de datos sensibles (generalmente de trabajo).

4.3. Instalando FreeSwan en el kernel de Linux

Como ya hemos dicho anteriormente, usaremos una implementación libre de IPsec para Linux como es FreeSwan. Freeswan son un conjunto de utilidades para el uso de IPsec, parches para el kernel y módulos que nos permitirán dotar a la pila de protocolos de la posibilidad de crear túneles IPsec.

La versión utilizada para este ejemplo ha sido *FreeSwan 1.98*, ya tenemos la versión 1.99 y 2.00. La versión 1.99 mejora ligeros detalles sobre 1.98 mientras que la 2.00 todavía no está aconsejada para producción.

Esta versión empleada en este trabajo, permite implantar IPsec en la serie de kernels 2.2 y 2.4 de modo que difícilmente nos encontraremos un kernel que no podamos usar esta característica.

Por otra parte, y aunque en principio el hecho de parchear un kernel o añadir al código del kernel de Linux nuevas características, no es lo más grato sobretodo para el administrador más novel, el paquete con en el que se distribuye Freeswan nos permite realizar estas acciones de un modo sorprendentemente sencillo y nada traumático para el usuario.

Del mismo modo, además de sencilla, la distribución de FreeSwan está muy bien documentada. En el directorio `doc` que aparece en el directorio `freeswan-1.98` podemos encontrar una gran cantidad de documentación desde explicaciones de uso y conceptos generales de IPsec hasta implantación en detalle de FreeSwan.

La distribución de *freeswan* supone la existencia de las fuentes de un kernel de Linux en el lugar habitual `/usr/src/linux` y las herramientas y programas necesarios para compilar un kernel (gcc, bin86, make, etcétera).

Una vez que tenemos todo esto y dentro del directorio donde descomprimimos FreeSwan ejecutaremos como root lo siguiente:

```
make menugo
```

Este target de make realiza las siguientes acciones: añade las funcionalidades IPsec al kernel, arranca la configuración del mismo para que podamos ajustarla si es necesario y posteriormente compila el kernel e instala los módulos pertinentes.

Las opciones añadidas por FreeSwan en la sección *Networking options* son las siguientes:

```
<M> IP Security Protocol (FreeS/WAN IPSEC)
      --- IPSec options (FreeS/WAN)
      [*]    IPSEC: IP-in-IP encapsulation (tunnel mode)
      [*]    IPSEC: Authentication Header
      [*]       HMAC-MD5 authentication algorithm
      [*]       HMAC-SHA1 authentication algorithm
      [*]    IPSEC: Encapsulating Security Payload
      [*]       3DES encryption algorithm
      [*]    IPSEC: IP Compression
      [*]    IPSEC Debugging Option
```

Estas opciones están correctas por defecto y hacen referencia al tipo de algoritmo de autenticación y modo de funcionamiento general de FreeSwan,

no se aconseja su cambio si no es por un administrador avezado que conozca a fondo el funcionamiento de FreeSwan.

Posteriormente arrancaremos con el nuevo kernel y cargaremos si es preciso el módulo de IPsec (únicamente existirá un modulo que es el que aparece marcado con la M por defecto). Una vez realizado todo esto, deberemos ejecutar dentro del directorio de FreeSwan:

```
make install
```

Para que instale en el sistema las utilidades necesarias para usar las características que hemos activado en el kernel.

Podemos comprobar que la configuración de IPsec es correcta con el siguiente comando:

```
ipsec verify
```

Que nos informará si existe soporte de IPsec y si todos los demonios están corriendo correctamente. En este punto tendremos nuestro sistema dispuesto para la configuración específica de IPsec.

4.4. Configurando IPsec/FreeSwan

Llegados a este punto es necesario tener conocimiento de la mecánica que realiza FreeSwan para el establecimiento de los túneles. FreeSwan enlaza un

interfaz de red virtual `ipsecX`, al interfaz de red con el que desee establecer el túnel y sobre el que ipsec rutará el tráfico que ha de ser encriptado.

Si ejecutamos la orden:

```
ip addr show
```

Podremos observar que aparte de los interfaces de red que teníamos hasta el momento, aparecen los interfaces `ipsecX` que hemos comentado anteriormente.

```
18: ipsec0: <NOARP> mtu 0 qdisc noop qlen 10
    link/ipip
19: ipsec1: <NOARP> mtu 0 qdisc noop qlen 10
    link/ipip
20: ipsec2: <NOARP> mtu 0 qdisc noop qlen 10
    link/ipip
21: ipsec3: <NOARP> mtu 0 qdisc noop qlen 10
    link/ipip
```

Por otra parte, recordar que estamos creando un túnel encriptado entre dos máquinas de modo que ambas máquinas han de activar el túnel para que el tráfico sea posible entre ambas máquinas.

Y ya por último, y antes de comenzar la configuración del túnel propiamente dicha, comentar que en las anteriores ejecuciones, se ha creado en `/etc/ipsec.secrets` una llave secreta que se usará para encriptar el tráfico.

El fichero de configuración que hemos de ajustar para configurar el túnel está en `/etc/ipsec.conf` y con el indicaremos los interfaces a los que queremos enlazar los interfaces virtuales y el túnel que queremos formar.

Como hemos hecho en los ejemplos anteriores explicaremos la configuración de un túnel con un ejemplo práctico como es el implantar un túnel directo entre dos máquinas con un interfaz *eth1* una de ellas y *eth0* la otra y con direcciones ip: `192.168.2.1` y `192.168.2.2`.

Mostraremos ahora el fichero de configuración `ipsec.conf` e iremos comentado cada una de las partes importantes del mismo:

```
config setup
interfaces="ipsec0=eth1"

klipsdebug=none
plutodebug=none

plutoload=%search
plutostart=%search
uniqueids=yes

conn %default
keyingtries=0
authby=rsasig


conn emain-hogwarts
left=192.168.2.2
```

```
right=192.168.2.1
auto=start
leftrsasigkey=0sAQPCaBngUbcCLMMQMcy4FDEegnrq+jGg8mIY470zDap
CgffqpkbNkZ350sHmF3jd4jfOcqv/I6FBD81pE8GFyC7yRQdl8M3gJ08BINz6obb94o
bzOxiF5Wk0YxSLEXXEd/N9ZUcmgD05Y3xOJH2jdRyuadZUJs31TrJwQ4freM9Mwj3LB
zpNhDgifjRRzOoTf660dVFslohEIQPqJ8rPLlxfFqWSWtVpGvGIL7BUQ3Aa3EvmLf2n
Zq88fEVZhQZiOphRsNeIdhtJQrwTGThNWWsbQABmToPt+cf2tAas6IpxO1TiflCUhvq
eIaB0d5eVnu26XpLP9Cx39pXo9itkwAgIAgdskgCNQSwDybVEx9oNJPpj

rightrsasigkey=0sAQPFleuDkijW3T6ospuuD74KG7dCaSQzqWNt6rMvUx
KWD+KFIAmVLR/BI+wrOytDfu8+bxBmVJrEdM1wj+Q6VjwtJzVoShjONJY51WKe0f+RD
JiB9I2wyCqAIGm/dhsbtYzcwTlGTDa77q7ulRsk/Blb8WmeUSn0lDRfoh+oH1nrJgjA
A3Ntw9KTQ6jCihZvA7Mt5EhATCBPTBGsrcUZdSTmmWTbjtPo/QUQgaH34pa4LclLDWz
Gbrq83Pmhfo9x0SmoIOH9NNt/Co9HNnY8WYN7aeI9bIOjbVjsIcM2xE8VTtvIQOlkur
c2++t/LCrHVGow8qKJP924hIINC2cBA84ihRZZAGHFwp5DwV765DAVgdlD
```

Repasemos las partes “no estandars” de la configuración que hemos adaptado para nuestro ejemplo.

En primer lugar vemos el parámetro `interfaces` que nos permite enlazar el interfaz virtual al interfaz de red con el que deseemos construir el túnel.

Posteriormente podemos ver la sección `conn` la cual hemos nombrado (por autodocumentación del fichero) con los nombres de ambos extremos del túnel y continuamos configurando este enlace, podemos observar los parámetros `left` y `right`, esto indicará las ips de ambos lados del túnel, podemos poner cualquiera de ellas como left o right, es indiferente, pero hemos de conservar este orden en ambos ficheros de configuración de las máquinas sobre la que se implanta el túnel.

El parámetro `auto` indicará que el enlace se levantará al comienzo de la conexión, también vemos los parámetros que quizás puedan llamar más la atención y son `leftasigkey` y `rightasigkey`. Estos parámetros indican las llaves con las que se encriptará tanto en la máquina "left" como en la "right" y se obtienen para cada una de las máquinas con el siguiente comando:

```
ipsec showhostkey
```

Cuyo resultado hemos de copiar exactamente en el fichero y cuyo orden ha de ser también idéntico como pasaba con la asignación de ips del túnel.

Como podemos deducir, el fichero de configuración de la máquina *par* únicamente cambiará en el interfaz escogido para enlazar el interfaz virtual IPSec.

Finalmente solo nos restará arrancar el servicio, lo cual, concretamente en Debian se realiza del siguiente modo:

```
/etc/init.d/ipsec start
```

Con esta orden se levantarán los interfaces y las rutas que antes apuntaban a *eth0* y *eth1*, se moverán a sus interfaces *ipsec0* de modo que el tráfico se rutará a través del interfaz virtual que se encargará de encriptar el tráfico en el túnel.

Esta es una sencilla configuración que no dista de ser la más común de las implantadas con esta herramienta y que posibilita la creación de una red privada virtual de un modo sencillo.

4.5. Algo más sobre IPsec/FreeSwan

Una vez hemos visto como configurar IPsec en nuestras máquinas Linux para conseguir túneles encriptados y habiendo avanzado de un modo eminentemente práctica y casi "a ciegas", es interesante observar que programas nos están dando este servicio.

FreeSwan lo forman básicamente dos *daemons*: *Pluto* y *Klips*.

Klips por su parte es el kernel que implementa los estándares IPsec, llevando acabo dos protocolos:

- **AH**(Authentication Header), el cual da un servicio de autenticación a nivel de paquete.
- **ESP**(Encapsulation Security Payload), el cual permite encriptar los paquetes además de autenticarlos.

Por su parte *Pluto*, es un servidor *IKE* de claves y de negociación. Que se encarga de negociar la conexión con el otro extremo del túnel completando así los estándares fundamentales IPsec.

Además de estos *daemons*, FreeSwan facilita alguna utilidad más, pero básicamente, la interacción posible con el usuario se realiza a través del comando `ipsec`, como ya hemos visto, que facilita cosas como mostrar la clave pública del servidor, verificar que ipsec esté funcionando correctamente en

nuestro sistema, rehacer la clave del servidor, mostrar parámetros de configuración y un largo etcétera.

Como podemos ver, es relativamente sencillo montar una implementación libre de IPsec para ordenadores Linux convirtiéndolos así en potentes routers de redes privadas virtuales a un coste realmente reducido.

5. Conclusiones finales

Con este trabajo se han pretendido dos objetivos:

Por una parte enseñar de manera práctica la implantación de parte del temario de la asignatura, ya que, nada hay más desazonador para el alumno realmente interesado en la telemática que "jugar" con simulaciones que, aunque interesantes, no le servirán para nada en su incorporación posterior al mundo laboral ni sacian sus ansias de ver la teoría plasmada en el mundo real. Al mismo tiempo, la implantación de estos servicios nos ha ayudado a fijar mejor las ideas de la propia teoría que se ha impartido en la clase.

Por otra parte este trabajo surge con la intención de demostrar que Linux es capaz de proveer servicios telemáticos de alto nivel como los que se imparten en cursos de redes de últimos cursos de ingeniería informática de un modo eficiente y robusto, lo que le permite ser un referente a la hora de la implantación de estos servicios a un bajo coste y grandes prestaciones.

Finalmente, los equipos usados para las pruebas fueron:

- **hogwarts**. AMD K7 2200XP+ (1800Mhz)
- 512MB de memoria RAM
- GNU/Linux (Debian Sid)
- Kernel 2.4.20
- Dos tarjetas ethernet *Realtek 8139* a 100Mb
- 80GB de disco duro.

y

- **emain**. AMD K6-II (350Mhz)
- 128MB de memoria RAM
- GNU/Linux (Debian Sid)
- Kernel 2.4.19
- Una tarjeta ethernet *Realtek 8139* a 100Mb
- 30GB de disco duro.

6. Referencias bibliográficas

6.1. IPv6

- Linux IPv6 Howto, *http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/Linux+IPv6-HOWTO.html*

6.2. QoS

- Artículo sobre QoS en Linux, José Luis Nogueira, *http://bulmalug.net/body.phtml?nIdNoticia=1727*.
- Linux advance routing & Trafic Control HOWTO, *http://www.lartc.net*.

6.3. IPsec

- Site del proyecto freeswan, *http://www.freeswan.org*.