

# Balancear conexiones a Internet howto

v. 1.1 (28/09/2004)

<b>Sección</b>	<b>Página</b>
1.- Introducción.....	2
2.- Puesta en marcha.....	2
2.1.- Configuración de las interfaces. ....	3
2.2.- Establecimiento de las rutas.....	4
2.3.- Activación del routing.....	6
3.- Ajustes y resolución de problemas.....	6
4.- El futuro.....	8
5.- Agradecimientos.....	8
6.- Copyrigh y licencia. ....	9

## 1.- Introducción.

La proliferación del uso de Internet es cada vez mayor y como consecuencia de ello la necesidad de ancho de banda a precios asequibles y con una alta disponibilidad. Por ello, es cada vez mas difícil encontrar una empresa u hogar que no disponga de conexión, con un ancho de banda cada vez mayor y en muchos casos con conexión permanente a la misma.

Sin embargo no en todos los casos podemos hacer frente a unas líneas de conexión a Internet llamemos "profesionales" y en otros casos lo que queremos es disponer de una redundancia en el enlace pudiendo tener conexión en el caso de que un proveedor tenga caído el servicio.

Imaginemos el caso de una mediana empresa, que dispone de una conexión a Internet basada en un ADSL de 2 Mbps. Con un uso elevado de la misma en la que además no quiere que las descargas de ftp de sus empleados acaben ralentizando la navegación de los directivos de la misma ;) aquí podemos encontrar que el balanceo entre 2 ADSL's puede ser una solución válida.

## 2.- Puesta en marcha.

La infraestructura de la que disponemos para poner en marcha el balanceo de conexiones es de un equipo Intel Pentium 4 con 512 Mb de RAM y 80 Gb. De disco duro. Equipado con 3 tarjetas de red Intel pro100. Sin embargo se puede utilizar un equipo de bastante menor entidad. Debe estar dimensionado al número de usuarios al que vaya a dar servicio.

Lo primero de todo es la elección del sistema operativo. En mi caso elegí Linux, como distribución Debian en su versión estable Woody. Después el kernel: yo bajé las fuentes del ultimo kernel de la rama estable, en este momento (sobre Agosto de 2003) era el 2.4.22.

Comenzamos por compilar el kernel; cada uno debe poner las opciones necesarias para su hardware, especialmente la tarjeta de red. En este caso, las 3 tarjetas de red son de la misma marca y solo necesitamos un driver en el núcleo el EtherExpressPro/100 (eepro100). Después vamos a necesitar activar como mínimo:

```
- IP advanced router -> IP: equal cost multipath
```

Además como vamos a aplicar reglas de firewall, debemos activar también la opción:

```
Networking options > Network packet filtering  
Networking options > Netfilter Configuration > IP tables support
```

Networking options > Netfilter Configuration > Packet filtering

Tras esto, ya tenemos nuestro kernel preparado para ser compilado. Comenzamos el proceso de compilación, no me voy extender demasiado sobre este proceso ya que existe multitud de documentación sobre ello y no es el objetivo del documento, estos son los pasos frecuentes para los kernel de la serie 2.4.x:

```
make dep
make bzImage
make clean
make modules
make modules_install
```

Ahora con nuestro nuevo kernel compilado, solo necesitamos ponerlo en el inicio para poder arrancarlo.

```
cp arch/i386/boot/bzImage /boot/vmlinuz-2.4.22
cd /
ln -s /boot/vmlinuz-2.4.22 vmlinuz-2.4.22
```

Y en el `/etc/lilo.conf` añade una entrada como esta:

```
image=/vmlinuz-2.4.22
label=Linux-2.4.22-lb
read-only
```

La etiqueta "Linux-2.4.22-lb" es solo un texto y vale cualquier cosa pero a mi me sirve para saber que version del kernel es y "lb" me recuerda que lo utilizo para hacer "load balancing". Reiniciamos la maquina y comprobamos que el proceso de arranque ha sido correcto.

Ahora necesitamos instalar el paquete `iproute`. En Debian se hace con el comando `apt-get install iproute`.

Vamos a dividir el proceso de puesta en marcha del enrutamiento en 3 pasos fundamentales y que nos van a permitir separar los procesos.

- Configuración de las interfaces.
- Establecimiento de las rutas.
- Activación del routing.

## 2.1.- Configuración de las interfaces.

Lo primero que vamos a hacer es borrar cualquier posible configuración anterior de las interfaces de red instaladas.

```
ip addr flush lo
```

```
ip link set lo down
ip addr flush eth0
ip link set eth0 down
ip addr flush eth1
ip link set eth1 down
ip addr flush eth2
ip link set eth2 down
```

Después, vamos configurándolas y levantándolas una por una:

```
Interfaz lo (Loopback)
ip addr add 127.0.0.1/8 dev lo
ip link set lo up
```

```
Interfaz eth0 (conexion LAN)
ip addr add 10.15.8.102/16 dev eth0
ip link set eth0 up
```

```
Interfaz eth1 (conexion ADSL 2 Mbps.)
ip addr add 10.69.69.1/24 dev eth1
ip link set eth1 up
```

```
Interfaz eth2 (conexion ADSL 512 Kbps.)
ip addr add 10.69.70.1/24 dev eth2
ip link set eth2 up
```

## 2.2.- Establecimiento de las rutas.

Debemos ocuparnos de limpiar también cualquier ruta existente en el equipo.

```
ip route flush default
```

Ahora, ajustamos las rutas que deben ser estáticas; me explicare con una par de ejemplos: Todos los paquetes que vayan redirigidos a la LAN, deben ser enrutados siempre por la misma interfaz o habrá determinadas cosas que podemos querer que vengan siempre por determinado interfaz. Cada LAN tendrá sus propias necesidades. En la mía, los servidores DNS no están en mi mismo segmento de red y tengo necesidad de enrutar las peticiones, específicas a los DNS por otro router distinto del de por defecto.

```
#Rutas estáticas.
route add -host 127.0.0.1/32 dev lo
route add -net 10.15.0.0/16 dev eth0
route add -net 10.69.69.0/24 gw 10.69.69.1 dev eth1
route add -net 10.69.70.0/24 gw 10.69.70.1 dev eth2

#Rutas a hosts especificos
route add -host 10.16.1.1 gw 10.15.8.10 dev eth0
route add -host 10.16.1.2 gw 10.15.8.10 dev eth0
route add -host 10.129.255.193 gw 10.15.8.10 dev eth0
```

Este es el punto culminante de nuestro montaje. Ahora es cuando indicamos al núcleo que los paquetes sin un destino concreto, es decir, los que van a entrar en la ruta por defecto, deben ser balanceados entre las líneas disponibles. Recordemos que nosotros lo estamos haciendo entre 2 líneas pero pueden ser n en función de las tarjetas que nuestro equipo tenga instaladas.

Para mayor versatilidad, y puesto que he tenido que hacer muchas modificaciones esta parte la he puesto en un script:

```
-----  
#!/bin/bash  
  
#Rutas  
echo "Ajustando rutas..."  
IF0="eth0"  
IF1="eth1"  
IF2="eth2"  
IP0="10.15.8.102"  
IP1="10.69.69.1"  
IP2="10.69.70.1"  
P1="10.69.69.8"  
P2="10.69.70.8"  
P0_NET="10.15.0.0"  
P1_NET="10.69.69.0"  
P2_NET="10.69.70.0"  
  
#Interfaz 0  
ip route add $P0_NET dev $IF0  
ip route add $P0_NET dev $IF0  
ip route add $P0_NET dev $IF0 src $IP0  
  
ip route add $P1_NET dev $IF1 table T1  
ip route add default via $P1 table T1  
ip route add $P2_NET dev $IF2 table T2  
ip route add default via $P2 table T2  
  
ip route add $P1_NET dev $IF1 src $IP1  
ip route add $P2_NET dev $IF1 src $IP2  
  
#LA REGLA MAGICA QUE BALANCEA ENTRE LAS 2 LINEAS  
ip route add default equalize scope global nexthop via $P2 dev  
$IF2 weight 1 nexthop via $P1 dev $IF1 weight 2  
  
ip rule add from $IP1 table T1  
ip rule add from $IP2 table T2  
-----
```

Para comprobar que los cambios han tenido efecto y el enrutamiento será redirigido hacia los 2 gateways podemos poner el comando:

```
ip route show
```

El cual nos tiene que dar una respuesta como la siguiente:

```
(...)  
default equalize  
    nexthop via 10.69.70.8 dev eth2 weight 1  
    nexthop via 10.69.69.8 dev eth1 weight 2
```

Nuestra regla mágica, consigue que la salida hacia internet por nuestros 2 proveedores sea "ecualizada". Además a la línea con mayor ancho de banda, le hemos asignado un peso 2 mientras que a la menor, le hemos asignado un peso de 1. En realidad y siguiendo con nuestro ejemplo deberíamos haber asignado un peso 4 a la línea de 2 Mbps. Ya que tiene 4 veces mas ancho de banda que una de 512 Kbps. Pero como muestra nos vale. Mi recomendación es observar la eficiencia en función del número de líneas y de los tráficos que vaya a soportar.

### 2.3.- Activación del routing.

Por último, vamos a activar el forwarding para que esto pueda funcionar. En realidad lo que hacemos es decirle a kernel que pase paquetes entre sus interfaces.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

A partir de aquí hay que ver como va a ser la salida de los clientes a Internet. Por ejemplo, yo tengo montado un proxy squid para que solo puedan salir a navegar por el puerto 80. Podemos sentarnos en frente de los routers y comprobar los LEDs de estado para ver su actividad. La alternativa que utilizo además de mirar a los routers ;) , es poner el paquete iptraf en la parte "General interface statistics", seleccionamos "all interfaces" y podemos comprobar cual es tránsito de los paquetes por cada uno de los interfaces. En nuestro ejemplo, debería ser: 2 peticiones por la línea de 2Mbps y la tercera por la de 512 Kbps y así sucesivamente. Aunque al principio los valores pueden desconcertarnos un poco, al cabo de un rato según el número de peticiones deben acabar siendo el doble por la interfaz eth1 que por la eth2 dado que le hemos asignado un peso del doble a una que a otra.

## 3.- Ajustes y resolución de problemas.

Vamos a hacer ciertas matizaciones acerca del funcionamiento del "load balancing":

No es estrictamente cierto que cada petición vaya a salir conforme a la regla establecida. La explicación es que el kernel guarda en una cache, la ruta seguida para alcanzar una determinada dirección. Esto quiere decir que si por ejemplo hacemos un ping a un sitio A luego otro ping a otra dirección B y luego de nuevo al sitio A el tercer ping debería salir por la línea de 512 Kbps. Sin embargo, la dirección del sitio A permanece en la cache y volveríamos a salir por la línea que salimos originalmente independientemente de que le correspondiese salir por otra. Para ver el estado de la caché pondremos el comando:

```
ip route show cache
```

O también en el fichero:

```
/proc/net/route
```

Esta caché creo que se refresca cada 300 segundos (5 minutos) pero puede ser limpiada manualmente si no nos interesa recordar estos valores:

```
ip route flush cache
```

También existe la posibilidad de escribir cualquier cosa en:

```
echo "algo" > /proc/sys/net/ipv4/route/flush
```

Uno de los puntos fuertes de este sistema, es la tolerancia a fallos que nos proporciona. Si algunos de los enlaces cayera, tan solo la petición que se estuviera efectuando por ese enlace se vería afectada dado que desde ese momento es descartada por el kernel y se volvería a intentar enviar algo por ella pasado 60 segundos. Este tiempo es ajustado por defecto pero puede ser modificado ajustándolo manualmente en las opciones del kernel. Por ejemplo, si queremos que se produzca una nueva comprobación cada 30 segundos:

```
echo "30" > /proc/sys/net/ipv4/route/gc_interval
```

Existen otros muchos (muchísimos) parámetros que podemos necesitar o simplemente querer cambiar. Así a voz de pronto, se me ocurre que podemos querer que nuestra máquina no responda a ping's, que no haga caso de las peticiones broadcast / multicast...pero en cuanto al comportamiento del router, nos pueden afectar entre otros:

```
/proc/sys/net/ipv4/route/gc_elasticity
```

Valores para controlar la frecuencia y comportamiento del algoritmo recolector de basuras de la caché de rutas. Indica los segundos antes de que el núcleo salte a otra ruta porque la anterior ha muerto. Por defecto 300.

`/proc/sys/net/ipv4/route/gc_timeout` Es el tiempo en el que el kernel declara una ruta como muerta. En función de este parámetro y `gc_interval` ajustaremos los tiempos para que el kernel deje de utilizar una ruta cuando esta cae o reintentará utilizarla.

`/proc/sys/net/ipv4/route/max_delay` Retrasos para vaciar la caché de rutas.

`/proc/sys/net/ipv4/route/max_size` Tamaño máximo de la caché de rutas. Las entradas viejas se purgan una vez que la caché alcance este tamaño.

Todos estos parámetros que afectan a la manera que el kernel tiene de trabajar con el enrutamiento es una de las partes menos documentadas del proyecto.

## 4.- El futuro.

Retomando lo que acabamos de ver acerca de los parámetros del kernel para ajustar el rendimiento del enrutamiento, el futuro debería pasar por documentar más ampliamente los parámetros del núcleo para poder optimizar el funcionamiento.

También estoy tentado de probar todo esto con los nuevos kernels de las series 2.6.X. Al compilar he encontrado opciones tan interesantes como:

```
IP: verbose route monitoring.  
TCP load balancing support.  
UDP load balancing support
```

¿Verdad que tienen muy buena pinta?

Hay un parche de Julian Anastasov en <http://www.ssi.bg/~ja/#routes-2.4> para los kernels 2.4.X he probado a aplicarlo y no he apreciado diferencias. Si alguien lo ha hecho, lo animo a que me lo cuente y lo añadiré a futuras versiones de este documento.

Como es de esperar no me comprometo a que haya futuras versiones y revisiones de este documento, pero en caso de que existiesen estas son las líneas maestras sobre las que quiero trabajar.

## 5.- Agradecimientos.

En lo personal, quiero agradecerle tanto a mi familia como a Arantxa la paciencia que tienen conmigo.

En lo profesional, mi primera dedicatoria tiene que ser a Antonio "chache" por empujarme en esto de la informática cuando todo eran cuevas arriba. Ahora que es un poco mas llano no puedo olvidarme de esas primeras clases practicas.

También quiero agradecer muy sinceramente a toda la gente que trabaja diariamente con licencias abiertas bien programando o bien escribiendo documentación que nos permite ampliar y compartir conocimientos y ponernos menos zancadillas. Gracias, a veces recupero algo de fé en el ser humano. ;)

## **6.- Copyright y licencia.**

Copyright (c) 2004 Francisco Javier Crespo Jiménez.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".