PRÁCTICAS DE TECNOLOGÍA DE REDES

(francisco.arguello@usc.es)

Departamento de Electrónica y Computación
Universidad de Santiago de Compostela
15782 Santiago (España)

22 de enero de 2020

Índice

1. Información de la red **1**

2. Emulación de red **5**

3. Subsistema de red **15**

4. Cortafuegos (iptables) **17**

5. VLAN **23**

6. Calidad de servicio **25**

7. Puentes y switches **34**

8. Multicast **36**

9. Enrutamiento avanzado **40**

10. Configuración de routers **49**

11. Dispositivos de red virtuales y túneles **62**

12. Diseño de la red de una empresa **68**

1. Información de la red

Linux dispone de una serie de comandos para mostrar la informacion de red:

- **Interfaces**. El comando `ifconfig` (o `netstat -i [-e]`) nos proporciona la dirección IP, difusión, máscara, características, estado, etc:

```
eth0  Link encap:Ethernet  HWaddr 00:04:76:99:09:97
      inet addr:192.168.20.30  Bcast:192.168.20.255  Mask:255.255.255.0
      inet6 addr: fe80::250:bfff:fedb:35e1/64 Scope:Link
      UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
      RX packets:5568 errors:0 dropped:0 overruns:0 frame:0
      TX packets:5546 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      RX bytes:590648 (576.8 Kb)  TX bytes:521044 (508.8 Kb)
```

  UP y RUNNING indican que el interface está activado y funcionando, BROADCAST y MULTICAST indican que admite difusión y multicast, TXQUEUELEN es la longitud que admite la cola de transmisión en número de paquetes y MTU (Maximum Transfer Unit) es el tamaño máximo de las tramas (1500 bytes es el tamaño máximo de las tramas Ethernet). El número de paquetes y bytes transmitidos y recibidos es acumulativo, es decir, cuenta desde la activación del interface. Tipos de errores en Ethernet:

  Collisions: se producen cuando dos o más interfaces intentan transmitir a la vez. Las colisiones pueden ser simples o múltiples: las múltiples son las que se producen repetidamente sobre varios intentos de transmisión del misma trama. Las colisiones son normales en el funcionamiento de la red Ethernet y típicamente están por debajo del 0.1 %.

  Errors: la trama se recibió completa pero el CRC es erróneo debido a ruido electrico, cables o conectores en mal estado, etc.

  Frame: la trama se recibió incompleta debido a colisiones o fallos de línea.

  Dropped: el interface se ha visto obligado a descartar tramas por falta de memoria, usualmente debido a un tráfico excesivo.

  Overrun: el interface se ha visto obligado a descartar tramas por falta de tiempo de procesamiento, usualmente debido a un tráfico excesivo.

  Carrier: el interface ha perdido la conexión con el hub o switch.

- **Rutas**. Las da el comando `route` (o bien `netstat -r [-n]`):

```
Destination  Gateway       Genmask       Flags Metric Ref Use Iface
192.168.20.0 *             255.255.255.0 U     0      0   0   eth0
127.0.0.0    *             255.0.0.0     U     0      0   0   lo
default      192.168.20.10 0.0.0.0       UG    0      0   0   eth0
```

Cada ruta consiste de una red destino (con su correspondiente máscara) y un router (gateway). Si la pasarela es * el interface está directamente conectado a la red. La métrica indica los saltos para llegar al destino. Los flags indican el estado de la ruta:

`U`	la ruta es factible	`G`	la ruta usa una pasarela
`D`	la ruta fue redirigida	`H`	el destino es un sólo ordenador

- **Estadísticas TCP/IP**. El comando `netstat -s` nos dice el número de paquetes recibidos, descartados, fragmentos reensamblados, etc.

```
Ip:
   105306229 total packets received      [paquetes recibidos en total]
   0 forwarded                           [paq. redirigidos]
   378 incoming packets discarded        [paq. recibidos y descartados]
   104080424 incoming packets delivered  [paq. recibidos y entregados]
   211020832 requests sent out           [paquetes requeridos a enviar]
   4520 outgoing packets dropped         [paq. que no se pudieron enviar]
   1043041 reassemblies required         [requerimientos de reensamblado]
   103053 packets reassembled ok         [paq. reensamblados OK]
   1483761 fragments created             [fragmentos creados]
Tcp:
   656545 active connections openings    [conexiones que se han abierto]
   1 passive connection openings         [conexiones pasivas, p.e. ftp]
   34 failed connection attempts         [intentos de conexion fallidos]
   4 connection resets received          [resets de conexion recibidos]
   6 connections established             [conexiones abiertas ahora]
   5788827 segments received             [segmentos recibidos]
   8756739 segments send out             [segmentos enviados]
   856 segments retransmited             [segmentos retransmitidos]
   34 bad segments received              [segmentos erroneos recibidos]
   6 resets sent                         [resets de TCP enviados]
Udp:
   323430 packets received               [paquetes recibidos]
   823 packets to unknown port received  [paq. recib. puerto desconocido]
   45 packet receive errors              [paq. recibidos con error]
   634343 packets sent                   [paquetes enviados]
```

- **Sockets abiertos**. Los da el comando `netstat -a [-t -u -n]`.

```
Proto Recv-Q Send-Q Local Address      Foreign Address      State
tcp        0      0 *:8000            *:*                  LISTEN
tcp        0      0 *:x11             *:*                  LISTEN
tcp        0      0 192.168.20.30:8000 192.168.20.33:327710 ESTABLISHED
tcp        0      0 192.168.20.30:8000 192.168.20.34:325633 FIN_WAIT2
udp        0      0 *:32768           *:*
```

LISTEN indica un socket de servidor abierto, es decir, que acepta conexiones, ESTABLISHED una conexión establecida, SYN_SENT que el socket están intentando realizar una conexión, TIME_WAIT, FIN_WAIT1, etc, que se encuentra en alguna fase de la desconexión.

- **Tabla ARP**. El comando `arp [-a]` da la correspondencia de dirección Ethernet/IP de máquinas de la misma red con las que ha habido conexión.

```
# NOMBRE (DIRECCION IP)     ETHERNET           DISPOSITIVO
gstic21  (192.168.20.31) at 00:04:76:99:08:FF [ether] on eth0
gsserver (192.168.20.10) at 00:04:76:98:CC:47 [ether] on eth0
```

  Cuando hagamos una transmisión a otro ordenador veremos que la tabla se amplía con el par dirección Ethernet/IP de ese ordenador.

- **Testeo de direcciones IP duplicadas**. Con el comando `arping`.

- **Modelo de Ethernet**. Lo da el comando `ethtool` *iface*:

```
Supported link modes:   10baseT/Half 10baseT/Full
                        100baseT/Half 100baseT/Full
                        1000baseT/Half 1000baseT/Full
Supported pause frame use: Symmetric Receive-only
Supports auto-negotiation: Yes
```

- **Tasa de transmisión**. Hay disponibles varias herramientas para determinar la tasa de transmisión:

  * `ifstat [-at]` (y también `tcpstat`) muestra la tasa por interface.

```
   Time            lo                 eth0
HH:MM:SS   KB/s in  KB/s out   KB/s in  KB/s out
09:39:09      0.00      0.00      0.18      0.26
09:39:10      0.00      0.00      0.10      0.12
```

  * `iftop` lo hace en función del destino en los últimos 2, 10 y 40 segundos.

```
ubuntu.local  =>  217-116-17-79.redes.acens  208b  208b  208b
              <=                              292b  292b  292b
```

  * `nethogs` muestra la tasa de transmisión por proceso.

```
2528  pepe  /usr/lib/firefox/firefox  eth0  0.039  0.051  KB/sec
?     root  unknown TCP                    0.000  0.000  KB/sec
```

- **Monitorizar el tráfico**. Con el comando `iptraf`:

```
TCP Connections (Source Host:Port)  Packets  Bytes  Flags  Iface
172.16.54.249:48660                 =     7   1612   --A-  eth0
199.16.156.201:443                  =     6   1519   -PA-  eth0
```

- **Testeo del ancho de banda de aplicaciones TCP/UDP**. Con el comando `iperf`.

- **Captura de paquetes**. Como sniffers disponemos de `tcpdump` en modo texto y `wireshark` (antes `ethereal`) en modo gráfico. Por su parte, `tcpflow`, permite reconstruir los datos de la capa de aplicación. `kismet` es un sniffer para transmisiones inalámbricas.

```
14:32:18.271598 IP decp0101.inv.usc.es > secus.usc.es:
      ICMP decp0101.inv.usc.es udp port 1335 unreachable, length 330
14:32:18.271904 IP decp0101.inv.usc.es.51047 > dns2.usc.es.domain:
      26735+ PTR? 249.54.16.172.in-addr.arpa. (44)
```

- **Búsqueda de texto en paquetes**. El comando `ngrep` permite buscar texto en los paquetes transmitidos mediante expresiones regulares. Por su parte `netsed` permite modificar los paquetes que son retransmitidos.

- **Escaneo de puertos**. `nmap` en modo texto y `zenmap` en modo gráfico.

```
PORT    STATE SERVICE
22/tcp  open  ssh
53/tcp  open  domain
631/tcp open  ipp
```

- **Testeo de rutas**. Disponemos del comando `ping` que testea un destino y `fping` que permite múltiples ping a los ordenadores de una red. Por su parte, `traceroute` muestra la lista de los routers intermedios.

```
 1  10.3.0.1 (10.3.0.1)  6.990 ms  7.096 ms  11.265 ms
 2  65.140.116.91.static.mundo-r.com (91.116.140.65)  12.936 ms * *
 3  212.51.33.218 (212.51.33.218)  14.971 ms  15.094 ms  15.215 ms
```

- **Testeo de servicios**. El comando `ssmping` prueba multidifusión, `echoping` prueba si un servidor está escuchando y `httping` prueba peticiones http.

- **Consultas del DNS**. Tenemos los comandos `nslookup`, `dig` y `host`.

- **Traer una página web**. Tenemos el comando `wget`.

- **Generar conexiones**. Tenemos el comando `nc` (netcat), que nos permite crear fácilmente sockets y realizar transmisiones. El servidor espera las conexiones y el cliente las solicita. A continuación lo que se escriba por teclado se transmitirá al otro extremo. Por ejemplo:

	TCP IPv4	UDP IPv4
Servidor	`nc -l 1234`	`nc -l -u 5300`
Cliente	`nc 127.0.0.1 1234`	`nc -u localhost 5300`

  Opciones: `-6` (utiliza direcciones IPv6), `-P` *proxy* (usa un proxy), `-p` *puerto* (selecciona el puerto origen), `-s` *IP_origen* (selecciona la IP origen).

- **wake-on-lan**. El comando `etherwake` permite despertar equipos.

2. Emulación de red

Usuaremos máquinas virtuales para estudiar diferentes configuraciones de red. Qemu/KVM es una aplicación que emula un ordenador completo, incluyendo el procesador y varios periféricos. La aplicación incialmente se llamó Qemu, pero al incluir recientemente la extensiones hardware de virtualización pasó a denominarse KVM. Además, usaremos este emulador cuando necesitemos trabajar con el usuario root.

2.1. Simulación de la red en Linux

La imagen que vamos a usar es `debian-sarge.img.gz` (versión KVM) o `debian-sarge.vdi.gz` (versión virtualbox). Necesitamos descomprimirla y generar dos copias. Por ejemplo, en KVM y para dejar el archivo comprimido sin modificar:

```
gzip -cd debian-sarge.img.gz > debian1.img
cp debian1.img debian2.img
```

A continuación elegir uno de los siguientes métodos de simulación:

2.1.1. KVM con disposivos tap

Este el método más testeado y recomendado para los portátiles. La conexión a la red es la más realista ya que se genera un puente en el ordenador y las máquinas virtuales se conectan mediante dispositivos *tap* (en los otros métodos la transmisión es a tráves del emulador).

1. Descargamos de la USC los archivos `inicia_qemu_tap.sh` y `qemu-ifup`, y les damos permiso de ejecución.

2. Necesitamos permisos de root para la creación del puente y los dispositivos *tap*, así que le damos permiso a KVM mediante `/etc/sudoers`. Tecleamos `visudo` e incluimos en este fichero la línea:

   ```
   ALL ALL=(ALL) NOPASSWD: /usr/bin/kvm
   ```

   Comprobamos que funciona tecleando:

   ```
   sudo kvm -net nic -net tap,script=qemu-ifup
   ```

   y con *ifconfig -a* (o con *ip -a*) debe aparecer el interface **br0** con dirección 192.168.0.1. Cerramos la ventana puesto que no hemos indicado las máquinas virtuales a ejecutar.

3. Por último, ejecutamos el script `inicia_qemu_tap.sh` que nos arranca las dos máquinas virtuales. El contenido de este archivo es el siguiente:

```
#!/bin/bash
sudo kvm -name "debian1" -hda debian1.img -vga cirrus -m 256 -device \
  ne2k_pci,netdev=net0,mac=00:01:01:01:01:01 -netdev tap,id=net0,script=qemu-ifup \
  -device ne2k_pci,netdev=net1,mac=00:02:02:02:02:02 -netdev \
  tap,id=net1,script=qemu-ifup -device ne2k_pci,netdev=net2,mac=00:03:03:03:03:03 \
  -netdev tap,id=net2,script=qemu-ifup &

sudo kvm -name "debian2" -hda debian2.img -vga cirrus -m 256 -device \
  ne2k_pci,netdev=net0,mac=00:04:04:04:04:04 -netdev tap,id=net0,script=qemu-ifup \
  -device ne2k_pci,netdev=net1,mac=00:05:05:05:05:05 -netdev \
  tap,id=net1,script=qemu-ifup -device ne2k_pci,netdev=net2,mac=00:06:06:06:06:06 \
  -netdev tap,id=net2,script=qemu-ifup &
```

Como puede observarse, cada vez que definimos un dispositivo de red (parámetro *-device*) le asignamos un dispositivo *tap* de conexión (parámetro *-netdev* que llama al script *qemu-ifup*).

Con este esquema de conexión podremos conectarnos a la máquina virtual sin necesidad de redirección (una vez configurados los interfaces), simplemente con:

```
ssh debian@192.168.0.2
```

2.1.2. KVM con conexión mediante sockets UDP

Este método y el siguiente son los recomendados si no es posible utilizar el método anterior. La conexión entre las máquinas se realiza mediante sockets UDP y la conexión al exterior se realiza a través de KVM (el método se denomina *user*).

Ejecutamos el script `inicia_qemu_udp.sh`, cuyo contenido es el siguiente:

```
#!/bin/bash
PORT=1201
kvm -name "debian1" debian1.img -vga cirrus -m 256 -device \
  ne2k_pci,netdev=net0,mac=00:01:01:01:01:01 -netdev \
  user,id=net0,net=192.168.0.0/24,host=192.168.0.1,\
  dhcpstart=192.168.0.2,dns=192.168.0.250,hostfwd=tcp::2222-:22 \
  -device ne2k_pci,netdev=net1,mac=00:02:02:02:02:02 -netdev \
  socket,id=net1,mcast=224.0.0.1:$PORT -device \
  ne2k_pci,netdev=net2,mac=00:03:03:03:03:03 \
  -netdev socket,id=net2,mcast=224.0.0.1:$PORT &

kvm -name "debian2" debian2.img -vga cirrus -m 256 -device \
  ne2k_pci,netdev=net0,mac=00:04:04:04:04:04 -netdev \
  socket,id=net0,mcast=224.0.0.1:$PORT -device \
  ne2k_pci,netdev=net1,mac=00:05:05:05:05:05 -netdev \
  socket,id=net1,mcast=224.0.0.1:$PORT -device \
  ne2k_pci,netdev=net2,mac=00:06:06:06:06:06 -netdev \
  socket,id=net2,mcast=224.0.0.1:$PORT  &
```

Donde PORT es el puerto usado en las transmisiones. Al ser multicast las transmisiones son recibidas por todos los hosts de la red, por lo que para evitar interferencias entre las diferentes máquinas cada usuario debe seleccionar un puerto diferente (al azar), que debe ser superior a 1024 para no necesitar los permisos del superusuario ni interferir con otras aplicaciones.

Además, hemos hecho una redirección desde el puerto 2222 del host al servidor ssh de la máquina virtual. Por tanto, una vez configurados los interfaces, nos podremos conectar a la máquina virtual con:

```
ssh -p 2222 debian@localhost
```

2.1.3. KVM con conexión mediante sockets TCP

Este método es similar al anterior con la diferencia que utiliza sockets TCP en vez de UDP. La conexión entre las máquinas se realiza mediante sockets TCP y la conexión al exterior se realiza a través de KVM (el método se denomina *user*).

Ejecutamos el script `inicia_qemu_tcp.sh`, cuyo contenido es el siguiente (PORT es el puerto usado en la conexión TCP):

```
#!/bin/bash
PORT=1201
kvm -name "debian1" debian1.img -vga cirrus -m 256 \
-net nic,vlan=0,macaddr=00:01:01:01:01:01 -net \
user,net=192.168.0.0/24,host=192.168.0.1,dhcpstart=192.168.0.2,\
dns=192.168.0.250,hostfwd=tcp::2222-:22 \
-net nic,vlan=1,macaddr=00:02:02:02:02:02 \
-net nic,vlan=1,macaddr=00:03:03:03:03:03 \
-net socket,vlan=1,listen=:$PORT &

sleep 1
kvm -name "debian2" debian2.img -vga cirrus -m 256 \
-net nic,vlan=1,macaddr=00:04:04:04:04:04 \
-net nic,vlan=1,macaddr=00:05:05:05:05:05 \
-net nic,vlan=1,macaddr=00:06:06:06:06:06 \
-net socket,vlan=1,connect=127.0.0.1:$PORT &
```

Hemos hecho una redirección de puerto para poder conectarnos a la máquina virtual como en el caso anterior.

2.1.4. Virtualbox

Alternativamente, las prácticas podrían realizarse con Virtualbox. Para ello necesitamos hacer lo siguientes ajustes:

1. **Máquina debian1**. Generamos una máquina debian de 32 bits denominada *debian1* a partir del archivo `debian-sarge.vdi` y 256 MB de RAM.

2. **Disco**. La imagen debian proporcionada trabaja con discos IDE (y no SATA). Por tanto en virtualbox en el menú de almacenamiento borramos el disco SATA y le añadimos un disco IDE.

3. **Red**. El interface conectado al exterior será de tipo *NAT*, mientras que los conectados a los interfaces de la red interna serán *internal network* con el nombre por defecto.

   Por defecto, las direcciones proporcionadas por NAT estarán en la red 10.0.x.0/24, siendo 10.0.x.15 la máquina virtual y 10.0.x.2 la pasarela. Podemos cambiarlas a 192.168.0.2 y 192.168.0.15, respectivamente, con el comando:

   ```
   VBoxManage modifyvm "debian1" --natnet1 "192.168.0.0/24"
   ```

	Qemu/KVM	Virtualbox
pasarela	192.168.0.1	192.168.0.2
máquina virtual	192.168.0.2	192.168.0.15

   Estos cambios en las direcciones del primer interface deberemos tenerlos en cuenta en todas las prácticas del curso. Podemos hacer también una redirección para conectarnos por ssh desde el puerto 2222:

   ```
   VBoxManage modifyvm "debian1" --natpf1 "guestssh,tcp,127.0.0.1,2222,,22"
   ```

4. **Máquina debian2**. Una vez generada la primera máquina virtual la clonamos para obtener una segunda, que denominaremos *debian2*.

5. **Ratón**. Si el ratón no se mueve o lo hace erráticamente, podemos eliminar la integración del ratón en virtualbox, pinchando con el botón derecho del ratón en los iconos de la parte inferior de la ventana.

2.2. Esquema de conexiones

2.2.1. Imágenes

En esta práctica simularemos dos máquinas virtuales con el S.O. Linux Debian Sarge que utilizaremos para estudiar distintas cuestiones de redes.

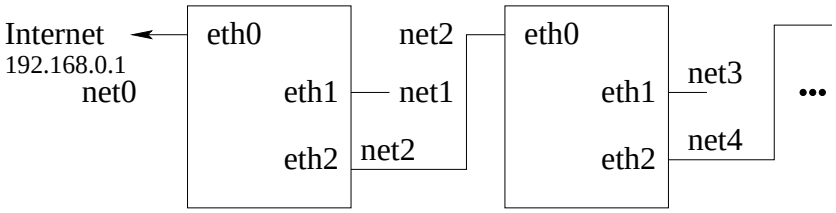
La primera máquina virtual carga la imagen del disco duro *debian1.img* y simula tres tarjetas de red con las direcciones MAC que le indicamos (el primer byte de la MAC debe ser par). La segunda simulación debe tener direcciones MAC distintas y un fichero de imagen de disco propio, puesto que cada emulación escribirá en su disco.

En las máquinas virtuales tenemos los usuarios *root* y *debian* ambos con la clave *debian*. Si entramos como usuario y después al pasar a root no nos deja

usar la pantalla gráfica, tecleamos como primer usuario **xhost +**. Disponemos del editor de textos gráfico *nedit* y el capturador de paquetes *ethereal* (antecesor de *wireshark*).

2.2.2. Parámetros de la red

El esquema típico de configuración que utilizaremos es el siguiente:



Sin embargo, por simplicidad, solo distinguiremos dos VLANs, la que contiene al interface que conecta al exterior y la que contiene al resto de los interfaces. Esto nos ahorrará tener que cambiar el esquema de conexión cada vez que queramos realizar una conexión diferente.

Concretando para el método de simulación de red que usa dispositivos *tap*,

1. La máquina virtual de la izquierda se comunica con el exterior a través del interface *br0* del PC real que actúa como pasarela. Averiguamos cuál es la pasarela tecleando `ifconfig br0` en el PC real. Si todo es correcto nos saldrá el interface *br0* con dirección IP 192.168.0.1.

2. La dirección de la tarjeta Ethernet `eth0` del PC virtual deberá pertenecer a esta red si queremos tener acceso a Internet. Por tanto la dirección de *eth0* deberá ser de la forma 192.168.0.x, x$\in [2, 254]$. En resumen:

Parámetro	Valor	Explicación
DHCP:	no	
Gateway:	192.168.0.1	La dirección de *br0*
Dirección IP:	192.168.0.x, x$\in [2, 254]$	En la misma red que *br0*
Máscara:	255.255.255.0	
Broadcast:	192.168.0.255	
Nameservers:	193.144.75.9, 193.144.75.11	Servidores de la USC
	8.8.8.8, 8.8.4.4	Servidores de Google

   En los otros métodos de emulación la transmisión se realiza a través de KVM, que es el que proporciona una pasarela interna.

3. De acuerdo con esto, configuramos los interfaces de red y la pasarela (con máscara de red y difusión por defecto):

```
ifconfig eth0 192.168.0.2
```

```
ifconfig eth1 192.168.10.2
ifconfig eth2 192.168.20.2
route add default gw 192.168.0.1
```

4. Para el DNS editamos el fichero de configuración `/etc/resolv.conf`. Usamos los servidores de nombres internos de KVM, de la USC o los de Google si estamos en casa:

   `nameserver 193.144.75.9` (o bien 8.8.8.8)
   `nameserver 193.144.75.12` (o bien 8.8.4.4)

5. Testeamos que la red funciona con el comando `host www.usc.es`. La transferencia de ficheros entre el PC real y el virtual puede realizarse usando el protocolo *sftp*. Por ejemplo, escribiendo en el PC virtual (si nuestro nombre de usuario tiene una arroba debemos ponerlo entre comillas):[1]

   `sftp "usuario@rai.usc.es"@192.168.0.1`.

6. **Apagado**. Al acabar, la imagen deberá cerrarse desde el menú de apagado de la máquina virtual, pues en caso contrario podría corromperse.

2.2.3. Grabación del estado de la máquina virtual

KVM permite generar snapshots de las máquinas virtuales para continuar posteriormente la ejecución en el punto en el que las hemos parado. La condición es que el formato de fichero de la máquina virtual los admita (por ejemplo, *qcow2*):

```
qemu-img convert -O qcow2 debian1.img debian1.qcow2
```

Podemos disponer de múltiples snapshots con su correspondiente ID:

1. Cuando estamos ejecutando la máquina virtual en KVM y queramos crear un snapshop, pasamos al modo monitor tecleando: CNTL-ALT 2.

   A continuación, paramos la emulación y creamos el snapshot:

   ```
   (qemu) stop
   (qemu) savevm ID
   (qemu) quit
   ```

2. Cuando queramos recuperar el snapshot, pasamos al modo monitor con CNTL-ALT 2 y tecleamos:

   ```
   (qemu) loadvm ID
   (qemu) cont
   ```

   Volvemos al modo de emulación tecleando CNTL-ALT 1.

[1] Adicionalmente podemos activar el escritorio gráfico remoto, para ello editamos en el PC virtual el fichero */etc/ssh/sshd_config* incluyendo la línea *X11Forwarding yes*, reiniciamos con `/etc/init.d/ssh restart` y nos conectamos al PC virtual con `ssh -X usuario@IP_virtual`.

2.3. Problemas

- **Proxy**. Para conectarse al exterior de la USC desde las máquinas virtuales de los PCs de cable del laboratorio podría ser necesario activar el proxy de la USC, bien con las opciones del navegador o con los comandos de terminal. En un terminal de debian-sarge, teclearíamos:

  ```
  export ftp_proxy=http://proxy2.usc.es:8080
  export http_proxy=http://proxy2.usc.es:8080
  export https_proxy=http://proxy2.usc.es:8080
  ```

  Para que esta configuración sea permanente, podemos escribir estas líneas en el fichero `.bashrc` de los usuarios *debian* y *root*.

  No se necesita en los portátiles ni tampoco si trabajamos desde casa.

  El navegador *firefox* tiene su propio menú de configuración de red en el que se puede poner el proxy.

- **No arranca kvm**. En el caso de que KVM no arranque, debido a que estamos dentro de una simulación virtualbox o no tenemos virtualización hardware, iniciar con:

  `kvm -no-kvm ....`

- **Ratón**. La velocidad del ratón se puede ajustar en el menú del entorno gráfico de la máquina virtual. Para capturar el ratón en Linux Mint hay que teclear CNT-ALT-G.

  Si el ratón no se mueve o lo hace erráticamente, podemos eliminar la integración del ratón en virtualbox, pinchando con el botón derecho del ratón en los iconos de la parte inferior de la ventana.

- **Interfaces de red**. En caso de que la máquina virtual no reconozca los interfaces de red, podemos probar indicándole a kvm otro modelo. La lista se obtiene con `-net nic,model=?`. Esta característica debe incluirse en todas las opciones que incluyan un *-net -nic*, esto es,

  ```
  kvm imagen.img -net nic,model=ne2k_pci -...
  kvm imagen.img -net nic,model=pcnet ...
  ```

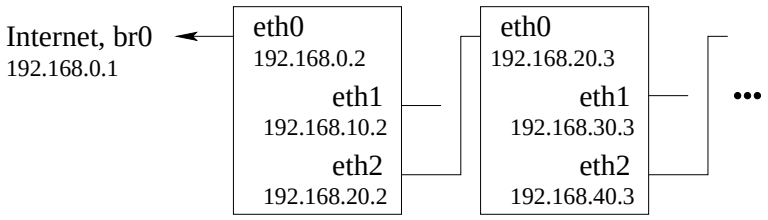
  Puede ser necesario cargar un módulo dentro de la máquina virtual con:

  `modprobe ne2k-pci` o `modprobe pcnet32`.

- **Formato raw**. Para evitar el mensaje *WARNING: Specify the 'raw' format explicitly to remove the restrictions*, podemos indicar el fichero de máquina virtual con la opción `-drive file=debian1.img,format=raw`.

2.4. Router

Un ordenador con Linux puede configurarse para trabajar como router si dispone de dos o más interfaces de red (varias tarjetas Ethernet, módem, etc). Además, de esta forma se dispone de todas las funcionalidades que posee Linux (cortafuegos, NAT, filtros por puerto o IP, VLAN, algoritmos de rutado, tunelización, etc). En esta práctica configuraremos en Qemu dos routers conectados como se muestra en la figura.



1. Configuramos cada uno de los interfaces con el comando `ifconfig`:

   `ifconfig` *interface IP* `netmask` *máscara* `broadcast` *difusión*

2. Configurar las rutas por defecto con el comando `route`:

   `route add default gw` *pasarela*

   Para el router de la izquierda la pasarela es 192.168.0.1, miestras que para el router de la derecha su pasarela es 192.168.20.2.

Además, para que el segundo router pueda conectarse al exterior, debemos configurar dos cosas en el router de la izquierda:

1. Para que Linux funcione como router, esto es, para que pase paquetes entre interfaces, hay que activar la característica de *ip_forward*. Esto se hace con el comando:

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

   El contenido del anterior fichero determina si la transferencia de paquetes entre interfaces está desactivada (valor 0) o activada (1). Alternativamente el cambio lo podemos hacer permanente incluyendo en el fichero */etc/sysctl.conf* la línea `net.ipv4.ip_forward=1`. Luego reiniciamos el ordenador o la activamos con el comando `sysctl -p`.

2. Las rutas que estemos usando internamente no serán accesibles al exterior puesto que al utilizar direcciones IP privadas, los routers de la USC las filtran. Por tanto, si queremos acceso a Internet en el router de la derecha, deberemos decirle al router de la izquierda que nos haga NAT:

   `iptables -A POSTROUTING -t nat -o eth0 -j MASQUERADE`

Adicionalmente podemos asignar más rutas, que es el objetivo básico de un router. El rutado puede ser estático o dinámico. Si todas las rutas van a ser estáticas podemos uasr el comando `route` para definirlas, pero si va a haber rutas dinámicas debemos usar algún protocolo de rutado. Ejemplos de uso del comando `route`:

`route add -host` *target* `gw` *pasarela*
`route add -net` *target* `netmask` *máscara* `gw` *pasarela*
`route add default gw` *pasarela*

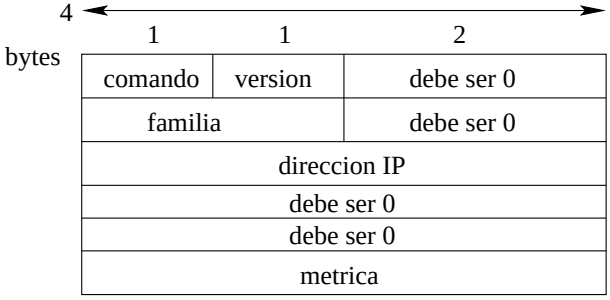
2.5. Protocolo de rutado RIPv1

La rutas dinámicas se generan con protocolos de rutado como RIP, OSPF, BGP, etc, en sus diferentes versiones como RIPv1, RIPv2, RIPng, etc. Si sólo estamos interesados en el protocolo RIPv1 (que es el más sencillo y con menos prestaciones), podemos utilizar el programa `netkit-routed`[2]. Si queremos un router más complejo, hay como alternativas los programas *gated*, *zebra*, *quagga*[3], etc, que permiten utilizar los protocolos RIP, OSPF y BGP en sus diferentes versiones. Estos routers más avanzados se verán en una práctica posterior.

RIPv1 se describe en el RFC 1058 (`http://www.faqs.org/rfcs/rfc1058.html`). Es un algoritmo de rutado de tipo *Vector de Distancias* en el cual los mensajes se envían sólo a los routers vecinos. RIPv1 se limita a redes cuya distancia más larga envuelve 15 routers. El algoritmo comprende los siguientes pasos:

1. Mantener una tabla de rutas en el sistema.
2. Periódicamente enviar un mensaje con toda la información de la tabla a los routers vecinos.
3. Cuando un mensaje llega desde un vecino, recalcular las rutas para comprobar si hay alguna de menor distancia.

RIP en un protocolo UDP que trabaja en el puerto 520. El formato de todos los mensajes se muestra en la siguiente figura. El conjunto de campos (dirección IP, 0, 0, métrica) indican un destino y una distancia y se repiten las veces necesarias.



[2]Disponible *netkit-routed* en `ftp://ftp.uk.linux.org/pub/linux/Networking/netkit/`.
[3]Disponible `http://www.gated.org`, `http://www.zebra.org`, y `http://www.quagga.net`.

Básicamente hay dos tipos de mensajes dependiendo del campo *comando*:
Solicitud. Para pedir información de rutas a los vecinos.
Respuesta. Contiene la información de rutas solicitada.

⇒ ENTREGA 1. Estudio del protocolo RIPv1

Mostrar RIPv1 funcionando, la captura de paquetes y las respuestas a las preguntas.

1. Configurar los dos routers como se explica en el apartado anterior. Comprobar que desde el router de la derecha funciona el comando *host www.usc.es*.

2. Visualizar las rutas que tenemos inicialmente y poner al programa *ethereal* (*wireshark*) a capturar paquetes en cada router.

3. Para iniciar RIP lo único que tenemos que hacer es teclear el comando `routed`[4] que se encarga de iniciar el proceso de sistema que gestiona RIP.

4. Después de algunos minutos funcionado, parar los capturadores de paquetes e identificar los mensajes RIP que se han estado enviando entre ellos. Observar con el comando *route* como se ha generando automáticamente la tabla de rutas.

5. Leer el contenido de los paquetes RIP capturados por *ethereal* y responder:

   a) ¿Cada cuánto tiempo se envían los mensajes RIP? No tener en cuenta que el qemu recibe paquetes triplicados o sextuplicados y fijarse en la secuencia de paquetes con el mismo origen.

   b) ¿Cuándo se envían las solicitudes y cuándo las respuestas? Esto es, deducir el esquema del protocolo solicitud-respuesta.

   c) ¿Qué métricas tienen los paquetes?

   d) A partir de las tablas de rutas iniciales de los hosts deducir el contenido de los paquetes de respuesta RIP iniciales (redes destino y métrica).

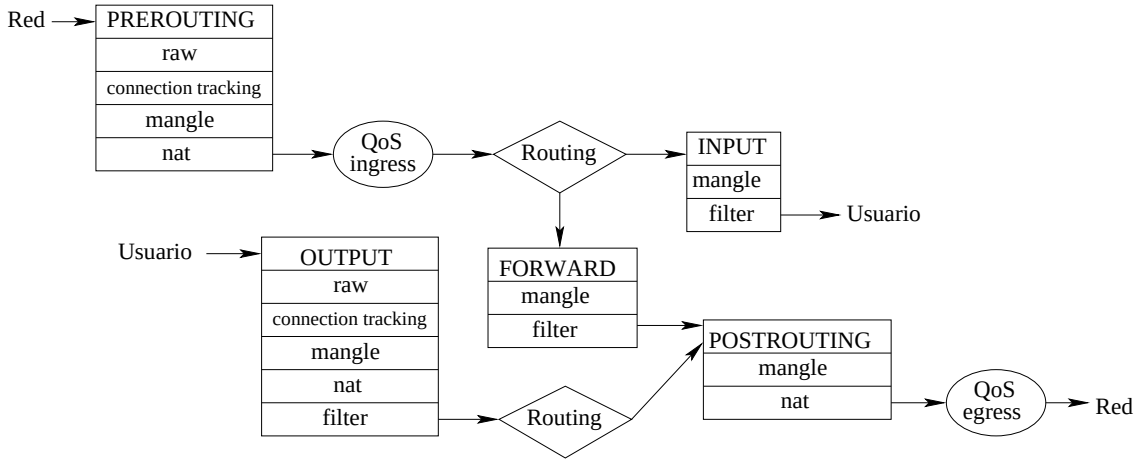
   e) A partir de los datos recibidos por el otro host deducir cómo se ampliará la tabla de rutas (red destino, métrica y pasarela).

   f) A partir de las tablas de rutas finales de los hosts deducir el contenido de los paquetes de respuesta RIP finales (redes destino y métrica).

[4] En Debian es necesario descargar el fichero *netkit-routed-0.17.tar.gz*, descomprimirlo, entrar en el directorio y teclear: `./configure`; `make`; `make install`. Esto ya está hecho en la imagen del disco duro proporcionada para simular con Qemu.

3. Subsistema de red

La siguiente figura muestra como está organizado el control de tráfico del S.O. Linux. Básicamente consta de 3 elementos:



1. Un sistema de tablas que permite el filtrado, traducción y modificación de paquetes en las distintas etapas de procesamiento. Las tablas se configuran con el comando *iptables*.

2. Un sistema de clasificadores y temporizadores de paquetes que proporciona calidad de servicio en las colas de entrada (QoS ingress) y de salida (QoS egress). Las colas se configuran con el comando *tc*.

3. Un sistema de rutado que decide el interface de salida a través del cuál será encaminado cada paquete. Se configura con el comando *ip*.

PREROUTING	paquetes entrantes al sistema
INPUT	paquetes destinados al sistema
OUTPUT	paquetes generados por el sistema
POSTROUTING	paquetes que salen del sistema
FORWARD	paquetes que atraviesan el sistema

- La tabla *raw* solo se usa para una cosa: marcar los paquetes que no queramos que sean rastreados por el sistema. No serán procesados por otras tablas.
- El módulo de seguimiento de conexiones (*connection tracking*) recuerda las conexiones realizadas, estados, etc, y según esto clasifica cada paquete.
- La tabla *filter* realiza el filtrado de paquetes de acuerdo a diferentes criterios (ip, puerto, etc) establecidos en la configuración.
- La tabla *mangle* permite marcar o modificar cada paquete.
- La tabla *nat* realiza la traducción de direcciones IP.

3.1. Variables de configuración de la red

Sysctl es un interface que permite modificar dinámicamente los parámetros de funcionamiento del kernel de Linux en relación con distintos subsistemas, incluyendo la red. En este caso, disponemos de las variables `/proc/sys/net/ipv4/*`. Existen decenas de estos parámetros, dentro de los cuales podemos destacar algunos relacionados con el funcionamiento normal de la red o con la seguridad:[5]

ip_forward	Permite retransmitir paquetes entre interfaces. BOOLEANO: 0=no (defecto), 1=sí. Por seguridad poner a 0 en los hosts y 1 en los routers.
ip_default_ttl	Valor por defecto del campo TTL (Time To Live). ENTERO: entre 0 y 255, por defecto 64.
route/max_size	Máximo número de rutas en la tabla de rutas. ENTERO. Se puede ampliar en los routers.
icmp_echo_ignore_all	Aceptar o ignorar los ICMP ECHO. BOOLEANO: 1=ignorar, 0=aceptar (defecto). Por seguridad podemos deshabilitar los pings.
icmp_echo_ignore_broadcasts	Aceptar o ignorar los broadcast ICMP ECHO. BOOLEANO: 1=ignorar (defecto) 0=aceptar.
igmp_max_memberships	Num. max. de subscripción a grupos multicast. ENTERO: defecto=20
proxy_arp	Permitir respuestas a solicitudes ARP desde fuera de la red. BOOLEANO: por defecto activado si algún interface lo soporta.
rp_filter	Rechaza los paquetes cuyo origen no se corresponde con una dirección alcanzable por el interfaz. 0=no se realiza comprobación. 1=rechazo si el interface no es el mejor path inverso (defecto). 2=rechazo si la dirección origen no es alcanzable desde el interfaz. Por seguridad conviene dejarlo activado.
accept_source_route	Aceptar paquetes con encaminamiento fuente. BOOLEANO: 1=sí (defecto en router), 0=no (defecto en host). Por seguridad conviene dejarlo deshabilitado en los hosts.
accept_redirects	Acceptar mensajes de redirección que nos envía un router. BOOLEANO: 1=sí, 0=no Por seguridad podría deshabilitarse.
log_martians	Reportar paquetes con direcciones imposibles. BOOLEANO: 1=sí, 0=no.
tcp_syncookies	Descartar conexiones antiguas (para evitar ataques SYN flooding). BOOLEANO: 1=sí (defecto), 0=no.
tcp_max_syn_backlog	Máximo número de conexiones TCP permitidas en cola. ENTERO: el número menor permitido es 128.

[5] `https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt`

4. Cortafuegos (iptables)

Iptables es una herramienta que forma parte del entorno *Netfilter*, incluido en el subsistema de red de Linux para trabajar con paquetes IP. Permite filtrar y modificar los paquetes IP, realizar traducciones de direcciones de red (NAT) o mantener registros de log. Sin embargo, no hace ruteo, por lo que se suele usar conjuntamente con *iproute*. Admite, entre otros, los siguientes parámetros (las opciones disponibles dependen de cada caso):

1. **Cadena** (opción `-A` para añadir, `-D` para borrar, `-L` para visualizar, `-F` para limpiar, `-P` política por defecto, `-N` cadena nueva): PREROUTING (paquetes entrantes al sistema), INPUT (paquetes destinados al sistema), OUTPUT (paquetes generados por el sistema), POSTROUTING (paquetes que salen del sistema) y FORWARD (paquetes que atraviesan el sistema).

2. **Tabla** (opción `-t`): filter (tabla de filtros, usada por defecto), nat (tabla de traducción de direcciones de red), mangle (tabla de alteración) y raw (para marcar paquetes que no serán controlados con estado de conexión).[6]

3. **Destino** (opción `-j`): ACCEPT (aceptar), DROP (descartar), REJECT (descartar y enviar un mensaje al origen del paquete), QUEUE (envío al espacio de usuario), RETURN (finalizar el procesamiento por la cadena), MARK (poner marca), SNAT (cambio de la dirección ip y puerto origen), DNAT (cambio de la dirección ip y puerto de destino), MASQUERADE (traducción automática), LOG (guarda información de los paquetes).

4. **Chequeo:** -i [!] *iface_entrada* (! indica negación), -o *iface_salida*, -p *protocolo* (ip, icmp, tcp, udp, etc), -s *ip_origen* (una red se indica añadiendo una máscara), -d *ip_destino*, --sport *puerto_origen* (varios puertos se separan por comas y puerto1:puerto2 indica un rango), --dport *puerto_destino*, -m multiport --ports *puerto* (origen o destino), --tcp-flags *banderas1 banderas2* (SYN, ACK, FIN, RST, URG, PSH, ALL y NONE), -m state --state (NEW,ESTABLISHED,RELATED,INVALID) -m mark --mark *marca*, -m owner --uid-owner usuario, -m icmp --icmp-type *tipo*, -m length --length *min:max*, -m time --weekdays *días* (por ejemplo, Sa,Su), -m time --timestart hh:mm[:ss] y -m time --timestop hh:mm[:ss] (definen un intervalo horario y también es posible especificar fechas de calendario).[7]

5. **Alteración:** --set-mark *marca* (para MARK), --to-source *ip* (para SNAT), --to-destination *ip* (para DNAT), --to-ports *puertos* (MASQUERADE), --set-tos *tipo.servicio* (para TOS), --ttl-set *tiempo.vida* (para TTL).

[6] Para filter las cadenas posibles son: INPUT,OUTPUT,FORWARDING, para nat: PREROUTING,OUTPUT,POSTROUTING y para mangle: PRE,I,O,F,POS.

[7] Algunas opciones dependen de plugins y su formato puede variar dependiendo de la versión.

4.1. Añadir reglas a una cadena

En este apartado usaremos la tabla por defecto, que es la tabla de filtros, usada para construir los cortafuegos. Para listar las cadenas disponibles usamos la opción `-L` (`-n` muestra direcciones IP en vez de nombres de host):

```
>>  iptables -L -n --line-numbers

Chain INPUT (policy ACCEPT)
num  target     prot opt source         destination

Chain FORWARD (policy ACCEPT)
num  target     prot opt source         destination

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source         destination
```

En este caso tenemos vacías las tres cadenas disponibles (INPUT, FORWARD y OUTPUT) y la política por defecto es aceptar todo tipo de paquetes. La política de cada cadena puede modificarse, por ejemplo,

```
>> iptables -P INPUT DROP
>> iptables -P INPUT ACCEPT
```

Añadamos tres reglas para filtrar los paquetes procedentes de conexiones del host 192.168.0.1, todos los paquetes del protocolo UDP y los paquetes TCP dirididos al puerto destino 23:

```
>> iptables -A INPUT -s 192.168.0.1 -j DROP
>> iptables -A INPUT -p udp -j DROP
>> iptables -A INPUT -p tcp --dport 23 -j DROP
>> iptables -L -n --line-numbers

Chain INPUT (policy ACCEPT)
num  target     prot opt source          destination
1    DROP       all  --  192.168.0.1     0.0.0.0/0
2    DROP       udp  --  0.0.0.0/0       0.0.0.0/0
3    DROP       tcp  --  0.0.0.0/0       0.0.0.0/0         tcp dpt:23
...
```

Vemos que la cadena INPUT tiene ahora 3 reglas. A partir de ahora podemos trabajar con números para borrar, insertar, reemplazar, etc. Hay que tener en cuenta que al borrar e insertar reglas, las restantes se renumeran, por ejemplo:

```
iptables -D INPUT 2
iptables -R INPUT 1 -d 192.168.0.1 -j DROP

num  target     prot opt source          destination
1    DROP       all  --  0.0.0.0/0       192.168.0.1
2    DROP       tcp  --  0.0.0.0/0       0.0.0.0/0         tcp dpt:23
```

Por último, eliminamos todas las reglas de la cadena con: `iptables -F INPUT`.

4.1.1. Orden de las reglas

Las reglas de una cadena se procesan una a una, por lo que el orden es importante. Por ejemplo, consideremos una regla que acepte los paquetes ssh y otra que descarte los paquetes de conexiones con destino 192.168.0.1. ¿Qué pasará cuando intentemos una conexión ssh a 192.168.0.1?

```
>> iptables -A OUTPUT -p tcp --dport 22 -j ACCEPT
>> iptables -A OUTPUT -d 192.168.0.1 -j DROP
>> iptables -L -n --line-numbers

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
1    ACCEPT     tcp  --  0.0.0.0/0       0.0.0.0/0         tcp dpt:22
2    DROP       all  --  0.0.0.0/0       192.168.0.1

>> ssh usuario@192.168.0.1 (Conexión aceptada)
```

En cambio, si las cambiamos de orden,

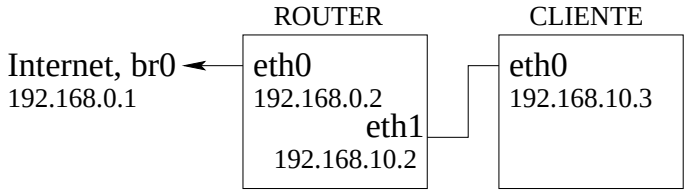
```
>> iptables -D OUTPUT 2
>> iptables -I OUTPUT 1 -d 192.168.0.1 -j DROP
>> iptables -L -n --line-numbers

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source          destination
1    DROP       all  --  0.0.0.0/0       192.168.0.1
2    ACCEPT     tcp  --  0.0.0.0/0       0.0.0.0/0         tcp dpt:22

>> ssh usuario@192.168.0.1 (Conexión rechazada)
```

Para las reglas ACCEPT/DROP los paquetes que las cumplen son aceptados/rechazados. No se siguen comprobando más reglas en la cadena actual. No obstante el paquete sigue atravesando el resto de cadenas y tablas.

La configuración actual puede guardarse y restaurarse con `iptables-save > fichero.txt` e `iptables-restore < fichero.txt`. Antes de pasar al siguiente apartado no olvidarse de borrar todas las cadenas de la tabla con la opción `-F`.

4.2. Operación con las distintas cadenas de una tabla

En este apartado usaremos el esquema de la siguiente figura. Veamos el funcionamiento de las tres posibles cadenas de la tabla de filtros: INPUT (paquetes destinados al sistema), FORWARD (paquetes que atraviesan el sistema) y OUTPUT (paquetes generados por el sistema).



Configuramos la primera emulación para que funcione como un router:

```
ROU>> ifconfig eth0 192.168.0.2
ROU>> ifconfig eth1 192.168.10.2
ROU>> route add default gw 192.168.0.1
ROU>> echo 1 > /proc/sys/net/ipv4/ip_forward
ROU>> iptables -A POSTROUNTING -t nat -o eth0 -j MASQUERADE
```

Configuramos el cliente de la forma usual:

```
CLI>> ifconfig eth0 192.168.10.3
CLI>> route add default gw 192.168.10.2
CLI>> host www.usc.es  (obtenemos respuesta)
CLI>> ping 192.168.0.2 (obtenemos respuesta)
CLI>> ssh 192.168.0.1 (obtenemos respuesta)
```

Añadamos al router un regla que impida las conexiones destinadas al sistema (INPUT). Sin embargo vemos que las conexiones que atraviesan el sistema (FORWARD) son permitidas:

```
ROU>> iptables -A INPUT -s 192.168.10.3 -j DROP
CLI>> ssh debian@192.168.0.2 (Conexión rechazada)
CLI>> ssh usuario@192.168.0.1 (Conexión aceptada)
```

Ahora en el router añadimos otra regla que impida las conexiones que atraviesan el sistema (FORWARD). Sin embargo las conexiones de salida (OUTPUT) son todavía permitidas:

```
ROU>> iptables -A FORWARD -s 192.168.10.3 -j DROP
CLI>> ssh usuario@192.168.0.1 (Conexión rechazada)
ROU>> ssh usuario@192.168.0.1 (Conexión aceptada)
```

Terminamos prohibiendo el último caso que nos queda:

```
ROU>> iptables -A OUTPUT -d 192.168.0.1 -j DROP
ROU>> ssh usuario@192.168.0.1 (Conexión rechazada)
ROU>> iptables -L -n --line-numbers

Chain INPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       all  --  192.168.10.3       0.0.0.0/0

Chain FORWARD (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       all  --  192.168.10.3       0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source               destination
1    DROP       all  --  0.0.0.0/0          192.168.0.1
```

Más adelante veremos que un usuario puede definir sus propias cadenas.

4.3. Tabla de traducción

En los apartados anteriores trabajamos con la tabla por defecto, la tabla de filtros. En este apartado vamos a estudiar la tabla nat. Vamos a utilizar el esquema de la figura anterior, pero antes de nada borramos todas las cadenas de la tabla de filtros: `iptables -t filter -F`. En la tabla nat tenemos las opciones MASQUERADE (cambia automáticamente las direcciones IP origen), SNAT (cambia las direcciones origen de forma manual) y DNAT (cambia las direcciones destino de forma manual).

MASQUERADE cambia las direcciones origen sin necesidad que se le indique ninguna dirección, pues coloca automáticamente la dirección válida del router. En el apartado anterior ya lo habíamos puesto para que el cliente pueda conectarse al exterior:

```
ROU>> iptables -A POSTROUNTING -t nat -o eth0 -j MASQUERADE
```

Podemos estudiar el funcionamiento capturando los paquetes de una conexión ssh 192.168.10.3 $\longrightarrow$ 192.168.0.1. En los paquetes que se transmiten en la parte que conecta el router con el exterior las direcciones son 192.168.0.2 y 192.168.0.1. Es decir, la dirección 192.168.10.3 se ha cambiado a 192.168.0.2.

SNAT necesita que se le indique la dirección nueva. Como es usual, supongamos que el interface eth0 tiene asignada la dirección 192.168.0.2. Vamos a cambiar la dirección de origen de los paquetes:

```
ROU>> iptables -t nat -A POSTROUTING -j SNAT -o eth0 --to-source 192.168.0.50
ROU>> ip a a 192.168.0.50/24 b 192.168.0.255 dev eth0
ROU>> iptables -t nat -L -n --line-numbers

Chain POSTROUTING (policy ACCEPT)
num  target     prot opt source      destination
1    SNAT       all  --  0.0.0.0/0   0.0.0.0/0     to:192.168.0.50
```

Adicionalmente hemos añadido una segunda dirección IP al interface eth0 para no tener problemas con los paquetes que llegan de vuelta. Con el capturador de paquetes podemos ver que los paquetes IP transmitidos llevan la dirección 192.168.0.50.

4.4. Tabla de alteración paquetes

Por último, vamos a trabajar con la tablas de alteración de paquetes (mangle). Con esta tabla podemos modificar ciertos campos de los paquetes IP (TOS y TTL) o bien ponerles marcas que después otros programas pueden examinar. Veamos dos ejemplos (solo necesitamos una máquina virtual):

```
>> iptables -t mangle -A OUTPUT -j TOS --set-tos 16
>> iptables -t mangle -A OUTPUT -j MARK --set-mark 5
>> iptables -t mangle -L -n --line-numbers

Chain OUTPUT (policy ACCEPT)
num  target     prot opt source      destination
1    TOS        all  --  0.0.0.0/0   0.0.0.0/0      TOS set 0x10/0xff
2    MARK       all  --  0.0.0.0/0   0.0.0.0/0      MARK set 0x5
```

El cambio en el campo TOS lo podemos ver con el capturador de paquetes examinando en una conexión de salida la cabecera de los paquetes. Los paquetes alterados tienen en el campo TOS 0x10 en vez del usual 0x00.

4.5. Creación de cadenas nuevas

En este apartado vamos a crear una cadena nueva a la tabla de filtros. Las cadenas nuevas se crean con la opción `-N` y se borran con `-X`. Supondremos que solo tenemos un acceso a Internet por eth0 y no queremos que nadie pueda acceder desde el exterior. Para ello prohibiremos las conexiones, salvo que estén inciadas o relacionadas con las generadas en el ordenador (por ejemplo, para ftp que usa una conexión diferente para control y datos).

```
# Cadena nueva que deja pasar por eth0 solo conexiones en curso o relacionadas
>> iptables -N block
>> iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
>> iptables -A block -m state --state NEW -i ! eth0 -j ACCEPT
>> iptables -A block -j DROP

>> Saltar a esa cadena en caso de conexiones procedentes del exterior
>> iptables -A INPUT -j block
>> iptables -A FORWARD -j block
```

⇒ ENTREGA 2. Ejercicios de Iptables

- **Prohibición de accesos**. Configurar iptables para el usuario *debian* no pueda acceder a las páginas web del servidor cv.usc.es, pero que este mismo usuario pueda acceder a esta máquina con otro tipo de protocolo (ping o ftp por ejemplo, OJO la USC puede filtrar algunos servicios) y también pueda conectarse a otros servidores web (probar con www.usc.es). El resto de los usuarios no tendrán restricciones de conexión. Toda la serie de condiciones puede escribirse en el mismo comando.

- **Configuración de rutas**. Configurar iptables para que los paquetes ssh vayan por rutas de alta seguridad, los restantes paquetes TCP (hacer uso del operador !) vayan por rutas de máximo ancho de banda, mientras que los paquetes UDP vayan por rutas de mínimo retardo. Comprobarlo con el capturador de paquetes.

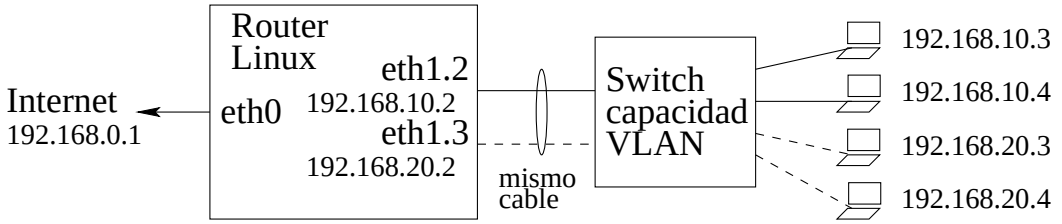
5. VLAN

Un ordenador con Linux puede utilizarse para definir VLANs (Virtual LANs). Estas son redes lógicas independientes construidas sobre el mismo hardware (mismos switches, routers y cables). Normalmente, si se conectan dos o más ordenadores al mismo switch pertenecerán a la misma LAN, pero con VLANs es posible separarlas. Un dispositivo de capacidad VLAN es visto como un conjunto de interfaces de red (interfaces virtuales), aunque en realidad sólo tenga un interface de red físico.

Es una técnica ampliamente utilizada por las empresas para aislar LANs y dar distintos tipos de acceso a distintos tipos de usuarios sin necesidad de duplicar el hardware.

Configuración

Para la configuración de las LANs virtuales en un router Linux utilizaremos la aplicación *vlan*[8], si bien en las nuevas versiones de Linux esto se puede hacer con el propio comando *ip*[9]. Utilizando el siguiente esquema creamos dos interfaces virtuales para el interface eth1. Le asignamos los números 2 y 3 pues la vlan1 suele reservarse para la propia gestión de las VLANs:



Empleados	192.168.10.0/24
Jefes	192.168.20.0/24

```
>> modprobe 8021q
>> ifconfig eth1 up
>> vconfig add eth1 2
>> vconfig add eth1 3
>> ifconfig eth1.2 192.168.10.2
>> ifconfig eth1.3 192.168.20.2
```

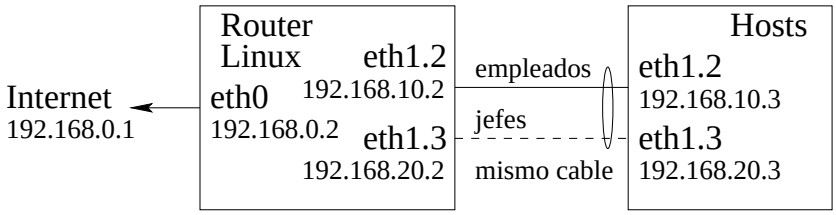
Para probar este esquema necesitaríamos un switch con capacidad VLAN, como los CISCO Catalyst.

[8] Disponible en `http://www.candelatech.com/~greear/vlan.html`. En la imagen de disco duro para Qemu está ya instalado.

[9] `ip link add link eth1 name eth1.2 type vlan id 2`.

⇒ ENTREGA 3. VLANs con permisos de iptables

En este ejercicio utilizaremos un segundo Qemu para simular otro dispositivo con capacidad VLAN y comprobar las transmisiones entre los dos.



Empleados	192.168.10.0/24
Jefes	192.168.20.0/24

1. Configurar las VLANs en el segundo Qemu de igual forma que en el primero, aunque con las direcciones IP específicas. Esta simulación emulará un host de tipo "empleado" (192.168.10.3) y otro de tipo "jefe" (192.168.20.3).

2. Configurar el router de la izquierda de forma que retransmita paquetes y permita acceder a Internet a los hosts.

3. Configurar las rutas por defecto en los dos Qemus. La ruta por defecto en el segundo Qemu se realizará a través de interface eth1.3.

4. En el router, usando *iptables* prohibir las conexiones nuevas procedentes de los hosts empleados destinadas al router. Se permitirán las conexiones iniciadas en sentido inverso (del router a los hosts empleados) y también las conexiones desde los hosts empleados a Internet. Para los hosts jefes no se pondrá ninguna restricción. Para ello, con iptables chequear el interface de entrada y hacer uso de `-m state` para impedir la creación de conexiones nuevas desde los hosts empleados. Esto permite reconocer si un paquete que viaja por eth1.2 destinado al router pertenece a una conexión iniciada por el router (permitida) o por un host empleado (prohibida).

5. Comprobar que hay acceso a Internet tanto desde el router como desde los hosts.

6. En el router comprobar que `ssh 192.168.10.3` y `ssh 192.168.20.3` son permitidas.

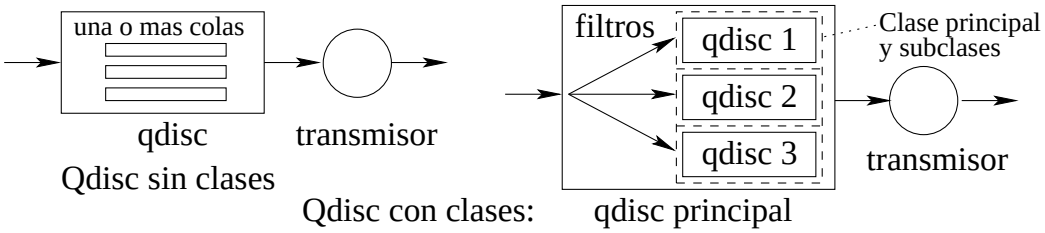
7. En los hosts, comprobar que `ssh 192.168.0.2` y `ssh 192.168.20.2` son permitidas

8. En los hosts, comprobar que las conexiones `ssh 192.168.10.2` son rechazadas.

6. Calidad de servicio

NOTA: Utilizar dos simulaciones de Qemu con tres tarjetas de red.

La calidad de servicio (QoS) está presente en Linux como parte del subsistema de red *iproute*[10]. Permite repartir el ancho de banda disponible en función de distintos criterios (IP, puerto, usuario, etc) y se basa en el control de tráfico en las colas de salida[11]. Los *filtros* reparten los paquetes entre las distintas *colas*, y estas deciden el orden de envío de los paquetes y también determinan el ancho de banda de la transmisión.



El comando básico es `tc`, que tiene la siguiente sintaxis:

```
tc [OPTIONS] OBJECT [COMMAND [ARGUMENTS]]
```

- OPTIONS. Comienzan por el caracter − y se pueden abreviar:

  `-s` o `-statistics` para obtener más información.

- OBJECT. Es el objeto que queremos gestionar.

  `qdisc` para la *disciplina de colas*, de la que existen varios tipos, siendo la más simple la de *pfifo*, en la cual los paquetes que llegan primero serán los transmitidos antes.

  `class` para las clases en las que se dividen algunas *qdisc*, para clasificar los paquetes usando filtros.

  `filter` para clasificar los paquetes y asignarlos a las correspondientes clases.

- COMMAND. El comando que se aplica al objeto.

  `add` para añadir una *qdisc*, *class* o *filter* a un interface.

  `del` para eliminar el objeto.

  `change` para cambiar atributos.

  `replace` equivale a *del* seguido de *add*.

NOTA: se usan las siguientes abreviaturas:

bytes (b, k, kb, m o mb)	bits (bit, kbit, mbit)
bytes/sg (bps, kbps, mbps)	bits/sg (bit, kbit, mbit)

[10] `http://linux-ip.net/articles/Traffic-Control-HOWTO/`, `http://www.opalsoft.net/qos/DS.htm` y `http://congreso.hispalinux.es/congreso2001/actividades/ponencias/eric/html/iproute.html`.

[11] De forma limitada se puede utilizar la qdisc `ingress` para las colas de entrada.

6.1. Disciplinas de cola sin clases (qdisc)

Al crear una disciplina de cola hay que indicar si se adjunta a la raíz de un dispositivo (`root`) o cuelgan de una clase indicando el `handle` esta última.

Las *disciplinas sin clases* tienen la sintaxis:

`tc qdisc add dev` *dev* `root` *qdisc qdisc-parameters*
`tc qdisc del dev` *dev* `root`

1. `pfifo / bfifo`: la disciplina más simple, contiene una sola cola de paquetes de tipo primero en entrar, primero en salir que acepta paquetes hasta alcance una determinada longitud límite (parámetro `limit`) medido en paquetes (`pfifo`) o en bytes (`pfifo`). Véase en el manual *man tc-pfifo* para más detalles. Por ejemplo:

```
>> tc qdisc add dev eth0 root pfifo help
>> tc qdisc add dev eth0 root pfifo limit 100
>> tc qdisc list
>> tc qdisc replace dev eth0 root bfifo limit 100k
>> tc qdisc del dev eth0 root bfifo
```

2. `red`: disciplina que elimina aleatoriamente paquetes para ajustarse al ancho de banda disponible. Cuando la longitud media de la cola está por debajo de `min` (medido en bytes) no se marca (desecha) ningún paquete; cuando excede de `min`, la probabilidad crece linealmente hasta que la longitud media de la cola alzanza `max`. Como en momentos puntuales la cola puede superar `max` se especifica el valor `limit` como longitud infranqueable; `avpkt` es el tamaño promedio de los paquetes y `burst` es el tamaño de las ráfagas (medido en paquetes) siendo un buen valor (2*min+max)/(3*avpkt). Ver en el manual *man tc-red* para más detalles. Por ejemplo:

```
>> tc qdisc add|replace dev eth0 root red min 10k max 80k limit 100k
   burst 30 avpkt 1000
```

3. `tbf`: disciplina que deja pasar paquetes hasta una determinada tasa, pero con la posibilidad de aceptar ráfagas cortas que sobrepasen ese límite; en otras situaciones se empieza a desechar paquetes. Es la disciplina sin clases más usada para limitar el tráfico de red en un interfaz. Algunos parámetros son: `rate` la tasa de transmisión, `limit|latency` que indica la longitud de la cola o el tiempo máximo que puede estar un paquete en la cola y `burst` el tamaño de las ráfagas puntuales (medido en bytes). Véase en el manual *man tc-tbf* para más detalles. Por ejemplo:

```
>> tc qdisc add dev eth0 root tbf help
>> tc qdisc add|replace dev eth0 root tbf rate 220kbit limit 100k burst 3k
```

4. `pfifo_fast`: disciplina compuesta por 3 colas FIFO donde los paquetes se clasifican en la correspondiente cola basándose en el campo TOS (type of service) o en su prioridad asignada por el programa. Primero se envían los paquetes de la cola 0 (TOS mínimo retardo), después los de la cola 1 (normal o TOS máxima fiabilidad), y por último, los de la cola 2 (TOS máxima capacidad). Es la disciplina que se usa por defecto si no se especifica otra y no es configurable. Por ejemplo:

```
>> tc qdisc del dev eth0 root
>> tc qdisc list
```

5. `sfq`: disciplina que otorga una cola FIFO para cada flujo, los cuales transmiten equilibradamente por turnos. Un flujo es una secuencia de paquetes que tienen suficientes parámetros como para distinguirlos, por ejemplo, la dirección de origen y de destino de los paquetes, así como los puertos. Los parámetros son: `quantum` el número máximo de bytes que una cola enviará antes de pasar a la siguiente y que, por defecto, es un paquete completo (1514 bytes en Ethernet) y `perturb` el tiempo (medido en segundos) en el cual se reinician las tablas. Véase en el manual *man tc-sfq*. Por ejemplo:

```
>> tc qdisc add|replace dev eth0 root sfq quantum 1514 perturb 15
```

6.2. Disciplinas con clases

Cuando un paquete llega a una qdisc con clases es clasificado en una clase dependiendo de los filtros que se hayan definido y va a parar a una qdisc secundaria. Cada elemento de la jerarquía se identifica con un handle/classid/flowid de forma `num1:num2`. El esquema típico de disciplinas, clases y filtros es el siguiente:

1. La qdisc principal se adjunta al *root* del dispositivo con *handle* 10:0
2. La clase principal tiene *parent* 10:0 con *classid* 10:1
3. Las subclases tienen *parent* 10:1 con *classid* 10:100 y 10:200
4. Las qdisc secundarias tienen *parent* 10:100 y 10:200 y su *handle* puede ser cualquiera.
5. Los filtros tienen *parent* 10:0 con *flowid* 10:100 y 10:200

Los filtros más comunes son *u32* y *fwmark*. El primero clasifica los paquetes basándose en los campos de la cabecera de paquete (ip src en el ejemplo), mientras que el segundo usa las marcas colocadas por *iptables* (marca 4 en el ejemplo):

```
>> tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip
   src 192.168.0.5 flowid 10:100
>> tc filter add dev eth0 parent 10:0 protocol ip prio 1 handle 4 fw flowid 10:100
```

Las disciplinas con clases más utilizadas son `prio`, `htb` y `cbq`.

1. `prio`: disciplina muy simple que contiene un número arbitrario de clases de distinta prioridad. El número de clases de la qdisc es configurable, aunque siempre mayor o igual a tres. Las clases son establecidas automáticamente, así que no es necesario teclear ningún comando para crearlas. Las clases son atendidas en orden de prioridad empezando por la de numeración más baja. Más información en el manual *man tc-prio*.

   ```
   tc qdisc add dev eth0 root handle 10:0 prio bands 3
   ```

   Las clases se numeran automáticamente 10:1, 10:2 y 10:3, a las cuales hay que asignarles las qdisc secundarias.

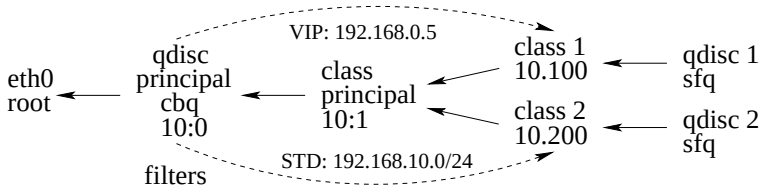
2. `htb`: sencilla disciplina que permite repartir el ancho de banda de transmisión. El parámetro *rate* indica el ancho de banda a la que la clase y sus hijos pueden transmitir, que puede superarse si hay ancho de banda disponible en la clase padre, mientras que *ceil* es infranqueable tanto para la clase como para los hijos. Las clases con *prio* menor son procesadas antes. En el manual, *man tc-htb*.

   ```
   >> tc qdisc add dev eth0 root handle 10:0 htb
   >> tc class add dev eth0 parent 10:0 classid 10:1 htb rate 5mbit
   >> tc class add dev eth0 parent 10:1 classid 10:100 htb rate 3mbit ceil 4mbit prio 1
   >> tc class add dev eth0 parent 10:1 classid 10:200 htb rate 2mbit ceil 3mbit prio 2
   >> tc class add dev eth0 parent 10:1 classid 10:300 htb rate 1mbit ceil 2mbit prio 3
   ```

3. `cbq`: disciplina que es la más compleja disponible y que permite configurar numerosos parámetros. Véase *man tc-cbq* y el siguiente ejemplo práctico.

Ejemplo de uso de la disciplina con clases `cbq`

Supongamos un ISP que necesita repartir el ancho de banda y que dispone de un router Linux con un interface *eth0* que está conectado a la red de clientes. El ancho de banda disponible es de 10 Mbps, que vamos a repartir en dos clases: "VIP" de 8 Mbps y "STD" de 2 Mbps.



- Eliminamos toda posible definición anterior de disciplinas, filtros y clases:

  ```
  >> tc qdisc del dev eth0 root
  ```

- Vamos a definir la disciplina `cbq` para *eth0*. La sintaxis la podemos consultar con *man tc-cbq*. En el comando indicamos que cuelga de `root`, le asignamos el `handle` *10:0* (también podemos escribir *10:*), le damos 10 Mbit de ancho de banda y le indicamos un tamaño medio de paquetes de 1000 bytes:

```
>> tc qdisc add dev eth0 root handle 10:0 cbq bandwidth 10Mbit avpkt 1000
```

- Ahora generamos nuestra clase raíz de la que cuelgan las demás. La sintaxis la podemos también consultar con *man tc-cbq*. Con `parent` *10:0* indicamos que desciende de la *qdisc* y le asignamos la `classid` *10:1*. Con `bandwidth` indicamos el ancho de banda del interface y con `rate` el que va a usar esta clase, con `allot` especificamos el número máximo de bytes a desencolar en cada turno y que hacemos igual a un paquete completo (1514 bytes). Las clases con mayor prioridad (número más bajo) serán procesadas antes.

```
>> tc class add dev eth0 parent 10:0 classid 10:1 cbq bandwidth 10Mbit
   rate 10Mbit allot 1514 avpkt 1000 prio 8
```

- Generamos luego nuestra clase VIP: le dedicamos 8 Mbit e indicamos con `bounded` que no se puede superar aunque las otras clases dejen ancho de banda disponible. Hacemos lo mismo con los 2 Mbit de la clase STD:

```
>> tc class add dev eth0 parent 10:1 classid 10:100 cbq bandwidth 10Mbit
   rate 8Mbit allot 1514 avpkt 1000 bounded prio 5
>> tc class add dev eth0 parent 10:1 classid 10:200 cbq bandwidth 10Mbit
   rate 2Mbit allot 1514 avpkt 1000 bounded prio 5
```

- Hasta ahora hemos indicado cuáles son nuestras clases pero no qué disciplina gestionará las colas. En este caso, vamos a usar la disciplina `sfq` (ver el apartado de disciplinas sin clases), que no es totalmente imparcial, pero que funciona bien para un ancho de banda bastante elevado, sin cargar demasiado. El procedimiento es el siguiente:

```
>> tc qdisc add dev eth0 parent 10:100 sfq quantum 1514 perturb 15
>> tc qdisc add dev eth0 parent 10:200 sfq quantum 1514 perturb 15
```

- Sólo nos queda hacer una cosa: indicar con filtros qué paquetes pertenecen a cada clase. Lo hacemos en función de la dirección IP, que en nuestro caso es 192.168.0.5 para la clase VIP y 192.168.10.0/24 para la clase STD. No olvidarse que estamos limitando el tráfico de salida del interface *eth0*, por lo que ponemos *dst*.

```
>> tc filter add dev eth0 parent 10:0 protocol ip prio 25 u32 match ip
   dst 192.168.0.5 flowid 10:100
>> tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 match ip
   dst 192.168.10.0/24 flowid 10:200
```

- Podemos comprobar que hemos hecho las cosas correctamente:

```
>> tc -s qdisc show dev eth0
>> tc -s class show dev eth0
>> tc -s filter show dev eth0
```

⇒ ENTREGA 4. Testeo de la tasa de transmisión

En este ejercicio vamos a comprobar que la limitación de la tasa de transmisión realmente está funcionando. Lo vamos a hacer en un solo Qemu distinguiendo dos usuarios locales (por ejemplo, *debian* y *root*).

1. Usaremos una *qdisc* de tipo `prio`, cuyo único parámetro (además del handle) es el número de bandas, en nuestro caso 3 (usaremos solo 2 pero 3 es el mínimo que admite). A las bandas se les asignan automáticamente los números 1, 2, etc, de tal forma que primero se atiende a la banda 1, cuando no hay nada en la banda 1 se atiende a la banda 2, etc.

2. Es esta disciplina no es necesario definir clases, pues automáticamente cada banda se convierte en clase. Los *classid* coinciden con los números de banda. Por ejemplo, si la *qdisc* es *10:0*, las bandas/clases serán *10:1* y *10:2* (verlo listando las clases).

3. Para las dos disciplinas asociadas a las clases usaremos `tbf` con tamaño máximo de cola 1514 bytes y tamaño de las ráfagas 1514. Para la banda 1 fijamos una tasa de 40 kbps (kbit según la nomenclatura del comando) y para la banda 2 de 10 kbps.

4. Para distinguir a los dos usuarios locales usaremos *iptables*:

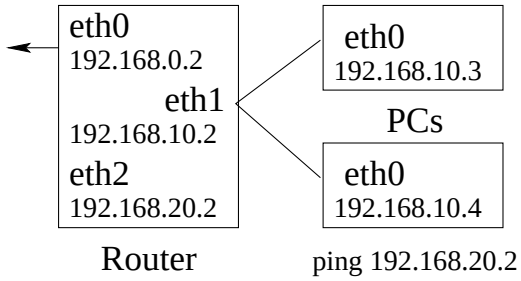
```
>> iptables -A OUTPUT -t mangle -m owner --uid-owner [usuario] -j MARK
   --set-mark [marca]
```

5. Usaremos filtros basados en las marcas de iptables con los siguientes parámetros: padre, protocolo IP, prioridad 1 para la banda 1 (2 para banda 2), `handle [marca] fw` y `flowid`.

6. Para comprobar la tasa, transferir un fichero de unos 100 kbytes desde Qemu al PC real con *sftp usuario@192.168.0.1* (OJO, solo hemos limitado el tráfico de salida, no el de entrada). Anotar la tasa y el tiempo de transmisión para los usuarios *debian* y *root*. Comprobar también que, cuando los dos usuarios intentan enviar a la vez, el de mayor prioridad transmite y el otro espera.

⇒ ENTREGA 5. Testeo del tráfico en un router

Ahora testearemos la QoS en un router y nos fijaremos en el tamaño de los paquetes. Usaremos dos Qemu con el siguiente esquema, con un Qemu para el router y un segundo Qemu que simulará los dos PCs, primero con una IP y luego con la otra:



Configuraremos la calidad de servicio en el interface *eth1* del router y permitiremos un tamaño máximo de 100 bytes para los paquetes enviados por el PC 192.168.10.3 y de 200 bytes para el PC 192.168.10.4.

1. Usaremos una disciplina principal de tipo `prio` y dos asociadas de tipo `bfifo`, que nos permiten limitar el tamaño de los paquetes.

2. Los dos filtros seran de tipo *u32* y estarán basados en `ip dst`, en ambos casos especificando la dirección del PC.

3. No olvidarse de poner la ruta por defecto en los PCs y de activar *ip_forward* en el router.

4. Podemos hacer las pruebas con el comando *ping*, pues este nos permite seleccionar el tamaño de los paquetes enviados. Hay que tener en cuenta que a los paquetes que genera ping hay que sumar los campos de la capa de enlace.

⇒ ENTREGA 6. Configuración de la red de una empresa

Nuestra empresa dispone de un servidor Linux por donde pasa todo el tráfico de Internet, telefonía IP y correo corporativo. Este servidor tiene dos interfaces de red: *eth0* conectado con un proveedor que nos proporciona acceso a a Internet con un ancho de banda de 40 kbps y *eth1* conectado con una red interna Fast Ethernet. Configuraremos la calidad de servicio en el interface *eth1* del router de la siguiente forma:

1. Vamos a definir 4 clases de ancho de banda.

   a) La Gerencia forma la subred 192.168.10.0/26. Le asignaremos la Clase I y le daremos el ancho de banda máximo disponible que es de 40 kbps.

b) El puerto usado por nuestro proxy de Internet es 8080. A esta aplicación le asignaremos la Clase II y un máximo de 20 kbps.

c) El resto de los usuarios pertenecen a la subred 192.168.10.64/26. Les asignaremos la Clase III y un máximo de 10 kbps.

d) Por último, nuestra sucursal es la subred 192.168.10.128/26. Le asignaremos la clase IV, y en este caso, los servicios de Internet y correo corporativo estarán limitados a 1 kbps.

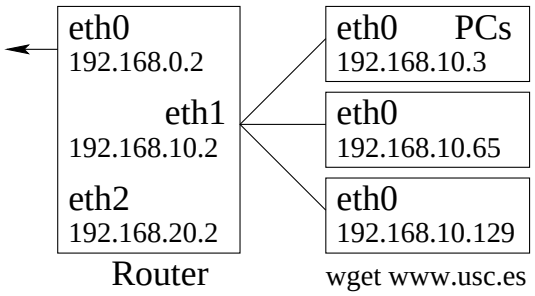
2. Para el interface raíz se usará la disciplina con clases `htb`.

   Para definir la *qdisc* no se necesitará ningún parámetro, mientras que para las cinco clases (principal + cuatro secundarias), los parámetros necesarios son `rate` para asignar el ancho de banda (que se puede superar en determinados instantes) y `ceil` para marcar el límite superior absoluto. La clase principal tomará la capacidad de la conexión y las secundarias tomarán las indicadas arriba. Las clase principal no tendrá prioridad, las clases I y IV tendrán prioridad 11, la clase II prioridad 12 y la clase III prioridad 13.

3. Para las cuatro disciplinas asociadas a las clases se usará `sfq`.

4. En cuanto a los filtros, para la clase II se usarán filtros de tipo *u32* con *ip dport* 8080 0xffff, y para el resto de las clases filtros basados en *ip dst*. Usar prioridad 10 para los filtros.

Para la comprobación del funcionamiento usaremos el siguiente esquema, con un Qemu para el router y un segundo Qemu que simulará tres PCs, cambiando en cada experimento la dirección IP:



Para realizar la comprobación:

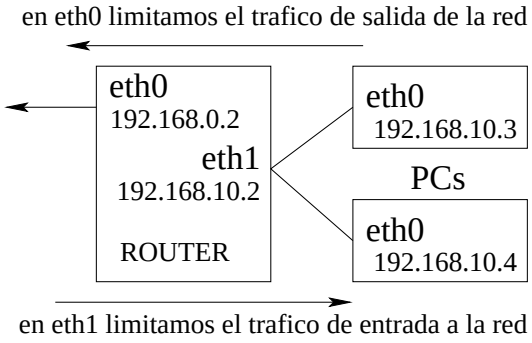
1. Asegurarse que el Qemu de la izquierda está funcionado como router, con la ruta por defecto por 192.168.0.1, con retransmisión de paquetes y nat.

2. En los PCs, cada vez que se cambie la dirección IP del interface, deberá configurarse la ruta por defecto a través del router.

3. Con el comando *wget* acceder a la página web de la universidad de santiago y comprobar el tiempo que tarda en recibirse, para cada una de las direcciones de los PCs.

6.3. Limitación del tráfico de entrada

Hasta ahora solo hemos considerado la limitación del tráfico de salida. La limitación del tráfico del tráfico de entrada se puede realizar de varias formas. Si la calidad de servicio se instala en un router con dos interfaces, uno para conectar con la red interna y el otro con el exterior, bastará con limitar el tráfico en ambos interfaces para limitar tanto el tráfico de entrada como de salida de la red.



Si la calidad de servicio la instalamos en un computador, podemos usar la disciplina de cola `ingress`. Es una *qdisc* de entrada un tanto limitada pero que permite ajustar la velocidad de transmisión de los diferentes tipos de transmisiones. El procedimiento considerando un único Qemu sería el siguiente:

1. Definir la *qdisc* de tipo `ingress` en el interface. No hay indicar que va a colgar de `root`, el único `handle` que admite es *ffff:0* y no tiene parámetros adicionales.

2. Definir los filtros que sean necesarios según los puertos, direcciones IP, etc, que queramos discriminar. Los parámetros que admiten los filtros son, por este orden: dispositivo, parent, protocolo, prioridad, filtro (por IP, puerto, etc), `police rate` (aquí indicamos la tasa de transmisión a la que queremos limitar), `burst` (ráfagas medidas en bytes), `drop` (para indicar que los paquetes se descartan, no requiere parámetros) y el `flowid`.

3. A modo de ejemplo, definir dos tasas de transmisión diferentes para la descarga de páginas web y de ficheros. Para el tráfico *ssh* (puerto 22) asignar una velocidad de 40 *kbps* y para el web (puerto 80) una velocidad de 10 *kbps*. Usar un `burst` de 2 *kbytes*.

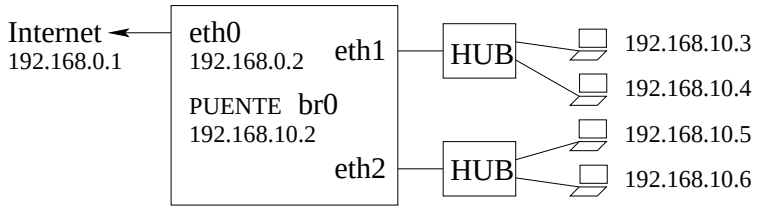
4. Probar la limitación de tráfico descargando del PC de sobremesa un fichero de 100 *kbytes* con *sftp* y una página web de la Universidad de Santiago con `wget` *http://www.usc.es*.

Existe también la posibilidad de usar la aplicación *squid* [12], que actúa a nivel de proxy y permite configurar por separado cualquier tipo de transmisiones, usuarios, con autorizaciones, ancho de banda, etc.

[12] `http://www.squid-cache.org/`.

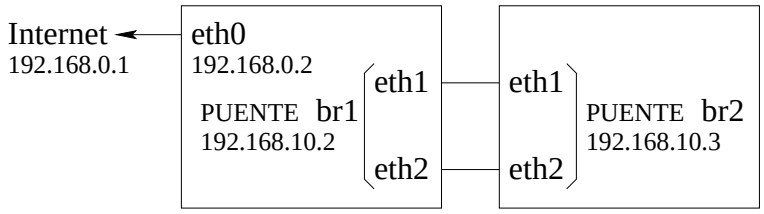
7. Puentes y switches

Un ordenador con Linux puesde configurarse para trabajar como un puente (o equivalentemente como un switch) si dispone de varios interfaces de red. Estos dispositivos trabajando en la capa de enlace aislan el tráfico de la LAN en tantas secciones como puertos dispongan (conmutan las transmisiones).



⇒ ENTREGA 7. Puente y protocolo de árbol de expansión

En esta práctica usuaremos dos puentes en un lazo cerrado para ver como se constituye el árbol de expansión IEEE 802.1d. Adicionalmente veremos como se construye la tabla ARP del ordenador y la tabla del puente. Usaremos dos simulaciones Qemu, una para cada puente.



El comando para generar el puente es `brctl` y seguiremos los siguientes pasos:

1. Creamos un interface simbólico para el puente, *br1*, en el Qemu de la izquierda y *br2* en el de la derecha.

   ```
   brctl addbr br1
   ```

2. Antes de seguir, lo mejor es activar el protocolo de árbol de expansión en los puentes, pues en caso contrario, podría haber una realimentación de envío de paquetes entre los dos puentes. Para ello debemos dar una prioridad a cada puente (por ejemplo, 1 y 2) y activar el árbol de expansión.

brctl stp <bridge> <state>	activa/desactiva árbol de expansión
brctl setbridgeprio <bridge> <priority>	fija la prioridad del puente
brctl setfd <bridge> <time>	fija el retado de retransmisión
brctl sethello <bridge> <time>	fija el intervalo de anuncios hello
brctl setmaxage <bridge> <time>	fija antiguedad máxima de mensajes
brctl setpathcost <bridge> <port> <cost>	fija el costo de los puertos
brctl setportprio <bridge> <port> <priority>	fija la prioridad de los puertos

3. A continuación le asignamos a cada puente los interfaces que van a colgar de él y le damos una dirección IP. Por ejemplo, para el puente de la izquierda:

   ```
   brctl addif br1 eth1
   brctl addif br1 eth2
   ifconfig br1 192.168.10.2
   ```

4. Configuramos los interfaces sabiendo que los puertos de un puente no tienen direcciones IP:

   ```
   ifconfig eth1 0.0.0.0 promisc
   ifconfig eth2 0.0.0.0 promisc
   ```

5. Configurar el tiempo de vida de las entradas del puente a 10 sg (ver la ayuda del comando `brctl`).

6. Completamos el esquema de prueba configurando en el Qemu de la izquierda el interface eth0, la ruta por defecto a través de la pasarela 192.168.0.1, activando la retransmisión de paquetes (ip_forward) y el NAT (con iptables). En el Qemu de la derecha ponemos la ruta por defecto a través de 192.168.10.2.

Ya tenemos todo preparado para estudiar el funcionamiento del puente. A continuación lanzar el capturador de paquetes.

1. En el puente de la derecha, mostrar la tabla del puente con el comando `brctl showmacs br2` y la tabla ARP del computador con el comando `arp -a`.

2. En el puente de la derecha, teclear `host www.usc.es`. A continuación, examinar la tabla del puente y la tabla ARP del computador. Volver a examinar las tablas después de un rato.

3. Explicar el significado de cada columna de la tabla del puente y de la tabla ARP, y cómo se obtienen los datos y cuándo se actualizan. Examinar los paquetes capturados y razonar con qué paquetes se han construido las tablas ARP y del puente.

4. Identificar los paquetes del protocolo de árbol de expansión y leyendo el manual explicar cómo se ha seleccionado la raíz del árbol.

5. Si se desea, se puede desactivar el árbol de expansión, para ver lo qué ocurre, aunque existe una probabilidad bastante grande de que las emulaciones se queden colgadas, debido al elevado número de paquetes retransmitidos.

8. Multicast

NOTA: La programación del multicast no requiere privilegios de administrador. En cambio, para la captura de paquetes, usaremos simulaciones de Qemu.

8.1. Direcciones multicast

Las direcciones multicast corresponden en IPv4 a la clase D, comienzan por 1110 y se subdividen en 3 rangos:

- *Reservadas*. El rango de direcciones 224.0.0.0-224.0.0.255 se reserva para los protocolos de rutado y de administración de la red local. Por ejemplo, 224.0.0.1 es el grupo de todos los host de una red, 224.0.0.2 es el grupo de todos los routers, 224.0.0.5 es el conjunto de todos los routers OSPF, etc.
- *Globales*. El rango 224.0.1.0-238.255.255.255 es gestionado por el organismo de asignación de direcciones de Internet. Las direcciones se asignan mediante petición y hacen posible el multicast en el global de Internet.
- *Privadas*. El rango 239.0.0.0-239.255.255.255 son direcciones multicast privadas: las transmisiones no saldrán de la red local.

El tiempo de vida de los paquetes (TTL) en el multicast IPv4 tiene un doble significado. Por un lado controla el tiempo de vida de un datagrama en la red, pero si además estamos trabajando con multicast, el TTL define el ámbito del datagrama, es decir, cómo de lejos llegará. Combinándolo con las direcciones:

TTL	direcciones	ambito
0		Nodo. No saldrá por el interface de red.
1	224.0.0.0-224.0.0.255	Enlace. No será encaminado por routers.
$2-31$	239.255.0.0-239.255.255.255	Local al departamento.
$32-63$	239.192.0.0-239.195.255.255	Local a la organización.
$64-255$	224.0.1.0-238.255.255.255	Global. Sin restricción.

8.2. Programación de multicast

En el host, hay que informar a la tarjeta de red de los grupos de multicast en los que estamos interesados, para que éste procese las correspondientes transmisiones. La programación es mediante sockets utilizando las funciones `setsockopt` y `getsockopt` para establecer o leer el valor de las opciones del multicast.

- La función `int getsockopt (int socket, int level, int optname, void *optval, socklen_t *optlen-ptr)` obtiene información sobre la opción `optname` al nivel `level` para el socket `socket`. El valor de la opción se almacena en el buffer al que apunta `optval`. Antes de la llamada, hay que

proporcionar en `*optlen-ptr` el tamaño de este buffer; la función devuelve el número de bytes que realmente se han almacenado en el buffer (la mayoría de las opciones interpretan este buffer como un único valor de tipo `int`. La función devuelve un 0 en caso de éxito y -1 en caso de fallo.

- La función `int setsockopt (int socket, int level, int optname, void *optval, socklen_t optlen)` se utiliza para establecer la opción `optname` al nivel `level` para el socket `socket`. El valor de la opción se pasa a través del buffer `optval` que tiene un tamaño `optlen`.

En el caso de las opciones de multicast, *level* es `IPPROTO_IP`, *opname* (opción) y *optval* (dato) se dan en la siguiente tabla y *optlen* se obtiene con `sizeof(`*dato*`)`.

Opción IPv4	Tipo de Datos	Descripción
`IP_ADD_MEMBERSHIP`	`struct ip_mreq`	Unirse a un grupo multicast.
`IP_DROP_MEMBERSHIP`	`struct ip_mreq`	Abandonar un grupo multicast.
`IP_MULTICAST_IF`	`struct ip_mreq`	Especificar un interface de red.
`IP_MULTICAST_TTL`	`u_char`	TTL para los paquetes multicast.
`IP_MULTICAST_LOOP`	`u_char`	Activar o desactivar el loopback para los mensajes multicast.

La estructura `ip_mreq` (que se define en el fichero de cabecera `netinet/in.h`):

```
struct ip_mreq {
  struct in_addr imr_multiaddr;    /* Dirección IP multicast */
  struct in_addr imr_interface; }; /* Dirección IP local del interface */
```

⇒ **ENTREGA 8**. Ej. 1 (programa C) y 2 (captura de paquetes).

Ejercicio 1. Vamos a enviar cada segundo un mensaje de texto a un grupo de multicast usando un programa de sockets UDP. Los receptores se unirán al grupo de multicast, leerán N mensajes y abandonarán el grupo.

1. El programa de envío es un programa de sockets normal que envía un mensaje a un determinado puerto (en nuestro caso 9930) y a una dirección de multicast (de acuerdo con la tabla, en nuestro caso 239.192.0.40). Enviar 10 mensajes con espera de un segundo entre mensajes, con la función *sleep*.

2. En el programa de envío establecemos con la función `setsockopt` un determinado tiempo de vida de los paquetes. Esto se puede hacer justo después de definir el socket.

3. El programa de recepcion también es un programa de sockets UDP normal que recibe en el puerto 9930 y con un lazo de recepción.

4. En el programa de recepción, después de definir el socket, deberá unirse al grupo (dirección) de multicast con `setsockopt`. Los parámetros son:

```
mreq.imr_multiaddr.s_addr=inet_addr("dirección_multicast_elegida");
mreq.imr_interface.s_addr=htonl(INADDR_ANY);
```

5. Optativamente, podemos establecer la propiedad que permite a múltiples programas usar el socket de recepción. Esto se hace con la función `setsockopt`, con *level* `SOL_SOCKET`, con *optname* `SO_REUSEADDR` y con *optval* un entero distinto de cero (hay que ponerlo como puntero).

6. Despues de recibir 10 mensajes, el host deberá abandonar el grupo de multicast y cerrar el socket.

7. Comprobar la recepción del mensaje de texto en distintos ordenadores de la red local. Comprabar también que el interface tiene asignadas las direcciones IP y ethernet multicast con `ip m l`.

8.3. Protocolo de Gestión de Grupos de Internet (IGMP)

IGMP es el protocolo que permite a los hosts comunicarse con los routers vecinos para las cuestiones relacionadas con el multicast. En concreto, hay dos tipos de mensajes IGMP

- *Informe* (Membership Report Message). Los hosts envían a sus routers estos mensajes para informarles de que desean unirse a un grupo, abandonarlo, o para cambios en el estado de recepción o de sus interfaces.

- *Consulta* (Membership Query Message). Los routers multicast envían regularmente (típicamente a intervalos de 1 minuto) una consulta a la dirección de multicast "todos los hosts". Cada host que desee ser miembro de uno o más grupo replica una vez por cada grupo en el que esté interesado. Cada respuesta se envía tras un intervalo de tiempo aleatorio para evitar aglomeraciones. Como a los routers no les importa cuántos hosts dentro de la misma red son miembros de un grupo, cualquier host que escuche en su red a otro host una solicitud al mismo grupo, se ahorrará su solicitud.

Los mensajes IGMP se envían sobre IP con el número de protocolo 2 y tipo de servicio 0. La cabecera IGMP (versión 3) de un mensaje de informe tiene el siguiente formato:

Tipo(8)	Reservado(8)	Suma de comprobacion(16)
Reservado(16)		Número de registros(16)
Registro de grupo(variable)		
...		

En donde *tipo* es `0x22` para un mensaje de informe (para los mensajes de consulta, que no vamos a ver, sería `0x11`). Puede haber varios registros de grupo, cada uno de los cuales contiene:

Tipo Reg(8)	Long.Datos(8)	Número de fuentes(16)
Dirección multicast(16)		
Dirección fuente unicast(16)		
...		
Datos(variable)		

- El tipo de registro puede indicar lo siguiente:

  `MODE_IS_INCLUDE` (1). Indica que los interfaces cuyas direcciones fuente se especifican NO leerán paquetes cuyo destino sea la dirección multicast que se especifica.

  `MODE_IS_EXCLUDE` (2). Indica que los interfaces de estas direcciones fuente SÍ leerán paquetes cuyo destino sea esta dirección multicast.

  `CHANGE_TO_INCLUDE_MODE` (3). Cambio a dicho modo.

  `CHANGE_TO_EXCLUDE_MODE` (4). Cambio a dicho modo.

  `ALLOW_NEW_SOURCES` (5). Indica que interfaces adicionales de un host desean unirse a un grupo al que ya pertenece otro interface del mismo host.

  `BLOCK_OLD_SOURCES` (6). Algunos interfaces del host saldrán del grupo.

Ejercicio 2: Ejecutar el programa de multicast en un Qemu (pueden ejecutarse servidor y cliente en la misma máquina). Debe estar configurado el interface y la ruta por defecto. Capturar los paquetes multicast con *ethereal* y responder:

1. Cuántos mensajes IGMP se han transmitido y de qué tipo?
2. ¿A qué dirección envía el servidor los mensajes multicast?, ¿Y a qué dirección se envían los mensajes IGMP?
3. ¿Qué tiempo de vida tienen los mensajes multicast?, ¿Y los mensajes IGMP?

8.4. Comandos relacionados con el multicast

- Mediante comandos podemos activar o desactivar la capacidad multicast de un interface (lo usual es que en los interfaces que lo admiten se active por defecto). También podemos activar el equivalente modo promiscuo de multicast:

  `ip link set` *iface* `multicast [on|off]`

  `ip link set` *iface* `allmulti [on|off]`

- El comando `ping 224.0.0.1` envía un eco al grupo de multicast de todos los host de la red y sirve para testear las capacidades multicast.

- Los comandos `ip m l` y `netstat -g` nos informan de los grupos de multicast a los cuales se ha unido nuestro host.

9. Enrutamiento avanzado

NOTA: Para ejecutar algunos comandos de esta práctica necesitaremos los permisos del usuario root, por lo que usaremos simulaciones de Qemu con 3 interfaces de red. Comenzaremos con la red sin configurar, pues esto lo haremos con los comandos explicados en esta práctica.

La mayoría de las distribuciones Linux y UNIX gestionan la red con los comandos clásicos *ifconfig*, *route* y *arp*. Sin embargo, en la actualidad Linux incluye un subsistema de red mejorado denominado *iproute2*[13]. Este nuevo subsistema de red tiene más posibilidades que los proporcionados por muchos routers, cortafuegos y dispositivos de control de tráfico comerciales de precio prohibitivo. Algunas de las características que porporciona *iproute* son:

1. Unifica los comandos relacionados con la gestión de la red (interfaces, rutas, tablas ARP, paquetes IP, etc). Los comandos clásicos *ifconfig*, *route* y *arp* están realmente implementados sobre *iproute2*.

2. Permite tablas de rutado múltiples que permiten hacer enrutamiento basado en múltiples criterios: usuarios, aplicaciones (puertos), direcciones MAC, direcciones IP, tipo de servicio, hora del día, contenido, etc.

3. Balanceo del tráfico entre varias rutas disponibles.

4. Creación de túneles IP (cifrados, IPv6 sobre IPv4, etc).

5. Permite calidad de servicio (control de tráfico) que permite la gestión y reserva del ancho de banda con varios métodos de clasificación, priorizado y limitación del tráfico tanto del entrante como saliente. La calidad de servicio la vimos en el capítulo anterior.

6. Restricción de los accesos y protección ante ataques DoS.

9.1. El comando básico `ip`

El comando básico para gestionar la red de forma unificada (interfaces, rutas, túneles, etc), se denomina `ip` (*/sbin/ip*) y tiene la siguiente sintaxis:

```
ip [OPTIONS] OBJECT [COMMAND [ARGUMENTS]]
```

- OPTIONS. Las opciones influyen comportamiento general del comando. Empiezan por el carácter − y se pueden abreviar. Algunas de ellas son:

  `-s` o `-statistics` para obtener más información.

[13] Consultar el Adv-Routing-HOWTO disponible en `http://www.linuxdoc.org` y la ponencia `http://congreso.hispalinux.es/congreso2001/actividades/ponencias/eric/html/iproute.html`.

`-f` o `-family` para especificar qué familia de protocolo usar: *inet*, *inet6* o *link* (cualquiera).

`-r` o `-resolve` para imprimir nombres DNS en lugar de direcciones de host.

- OBJECT. Es el objeto del que buscamos información o queremos configurar. Disponemos de los siguientes objetos, que también pueden ser abreviados:

  `link` o `l` para los interfaces de red.

  `addr` o `a` para las direcciones de los interfaces.

  `maddr` o `m` para las direcciones multicast.

  `neigh` o `n` para la tabla ARP.

  `rule` o `ru` para las reglas de la política de rutado.

  `route` o `r` para las tablas de rutas.

  `tunnel` o `t` para los túneles sobre IP.

- COMMAND. Es el comando que se aplica al objeto. Se puede abreviar:

  `add` o `a` para añadir un atributo a un objeto.

  `del` o `d` para borrar un atributo a un objeto.

  `flush` o `f` para limpiar los atributos del mismo tipo en un objeto.

  `set` o `s` para activar/desactivar una característica en un objeto.

  `show`, `list` o `l` para ver un objeto.

Algunos usos del comando `ip` son los siguientes:

1. Mostrar la ayuda. En casi cualquier momento podemos solicitar ayuda añadiendo `help`, por ejemplo, `ip h`; `ip l h`; `ip l s h`; etc.

2. Mostrar los parámetros del interface de red: `ip link list`. Podemos añadir `dev` *name* para especificar uno determinado y `up` para mostrar sólo los que están activos.

```
>> ip l l
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0c:6e:16:58:83 brd ff:ff:ff:ff:ff:ff
```

   La primera línea de cada entrada muestra el número único del interfaz, su nombre (que puede ser cambiado), así como información sobre el estado del interfaz (*qdisc* lo veremos en el apartado de calidad de servicio). La segunda línea da información sobre el tipo de interfaz de que se trata, la dirección hardware o de capa de enlace (que en Ethernet es la dirección MAC).

La opción `-s` nos permite ver estadísticas del interfaz. Los parámetros son similares al antiguo comando *ifconfig*.

```
>> ip -s l l
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0c:6e:16:58:83 brd ff:ff:ff:ff:ff:ff
    RX: bytes  packets  errors  dropped overrun mcast
    1789685066 1975826  99656   0       0       0
    TX: bytes  packets  errors  dropped carrier collsns
    1619835989 2304565  69      16      4       1762589
```

3. Modificación de los parámetros del interfaz: `ip link set`. Especificamos con `dev` *name* de qué interfaz se trata y algunas opciones disponibles son: `up/down` que activa/desactiva el interfaz, `name` *name* que cambia su nombre, `promisc [on|off]` que pasa al modo promiscuo y `mtu` *number* cambia el *Maximum Transmission Unit* (MTU). Por ejemplo,

```
ip l s dev eth0 up
ip l s dev eth0 promisc on
```

4. Gestión de las direcciones del interfaz: `ip addr`. Este comando permite listar las direcciones, añadir nuevas o borrarlas. Es importante destacar que en *iproute*, las interfaces físicas y las direcciones son totalmente independientes, es decir, que una interfaz puede tener varias direcciones sin necesidad de crear un alias como ocurría antes. Con `list` mostramos las direcciones, con `add` añadimos una nueva dirección y con `del` borramos una. Si el interface tiene una dirección antigua y queremos cambiarla por una nueva, primero deberemos borrar la antigua. Algunos ejemplos de uso son los siguientes:

```
>> ip a h
>> ip a a 193.144.84.172/24 b 193.144.84.255 dev eth0
>> ip a l
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:0c:6e:16:58:83 brd ff:ff:ff:ff:ff:ff
    inet 193.144.84.172/24 brd 193.144.84.255 scope global eth0
    inet6 fe80::20c:6eff:fe16:5883/64 scope link
>> ip a d 193.144.84.172 b 193.144.84.255 dev eth0
```

5. Gestión de la tabla ARP: `ip neigh`. Se le puede añadir `list` para mostrar la tabla ARP, `del` para borrar una entrada (realmente la entrada correspondiente no desaparecerá de inmediato, se quedará hasta que el último cliente que la usa la libere), `add` para añadir una nueva entrada o `change` para cambiar una entrada. Por ejemplo:

```
>> ip n l
192.168.2.52 dev eth0 lladdr 00:05:1c:01:5e:1b nud delay
>> ip n d 192.168.2.52 dev eth0
```

6. Túneles: `ip tunnel`. Visto en un apartado anterior.

9.2. Tabla de rutas múltiple

El comando `ip` permite gestionar tablas de rutas múltiples, algo nuevo en el subsistema de red de Linux. Hasta ahora teníamos una tabla de rutas única que indica qué red está conectada a un interfaz, cuál es la pasarela, etc. Con *iproute* podemos elegir la tabla de rutas a usar en base en múltiples criterios: usuarios, aplicaciones (puertos), direcciones MAC, direcciones IP, tipo de servicio, hora del día, contenido, etc.

El comando que gestiona las reglas de la política de rutado es `ip rule` (abreviatura `ip ru`). Inicialmente existen 3 tablas de rutas: el comando clásico *route* muestra la tabla *main*, mientras que las otras dos son nuevas. La tabla *local* es especial, no puede borrarse y tiene la prioridad más alta (0), se usa para direcciones locales y de broadcast. La tabla *default* esta vacía y se reserva para procesos de post-rutado (si las reglas anteriores no coinciden).

```
>> ip ru l
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

La tabla *main* especificará la ruta para la mayoría de los paquetes, pero nosotros podemos crear tablas de rutas adicionales para casos particulares. Por ejemplo, `from`: según el origen del paquete, `to`: según el destino del paquete, `iis`: según el interfaz de llegada, `tos`: según el valor del *type of service* (TOS) y `fwmark`: según el valor de marca del paquete puesto por *iptables*. En el caso de `fwmark` hay que usar *iproute* conjuntamente con *iptables* que nos provee de un sistema para marcar paquetes (lo veremos más adelante). Por ejemplo:

```
>> echo 100 Tabla1 >> /etc/iproute2/rt_tables
>> echo 101 Tabla2 >> /etc/iproute2/rt_tables
>> echo 102 Tabla3 >> /etc/iproute2/rt_tables
>> ip ru a to 192.168.10.0/24 table Tabla1
>> ip ru a from 192.168.0.35 table Tabla2
>> ip ru a fwmark 1 table Tabla3
```

Las tablas de rutas se gestionan con el comando `ip route` (abreviatura `ip r`). Por ejemplo, para la tabla *main*:

```
>> ip r a default via 192.168.0.1 dev eth0 table main
>> ip r l table main
192.168.0.0/24 dev eth0  proto kernel  scope link src 192.168.0.2
192.168.10.3 dev ppp0  proto kernel  scope link  src 192.168.10.2
192.168.20.3 dev ppp1  proto kernel  scope link  src 192.168.20.2
default via 192.168.0.1 dev eth0
```

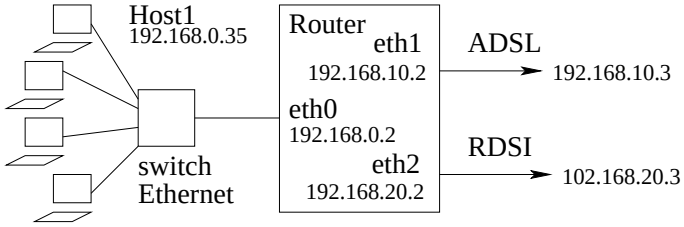
Hemos asignado la ruta por defecto a través de la pasarela 192.168.0.1. La primera línea de la tabla nos dice que a la red 192.168.0.0/24 se llega a través del interface *eth0* que tiene de dirección 192.168.0.2.

El contenido de una tabla de rutas puede vaciarse con la opción `flush`, por ejemplo, `ip r f table Tabla1`. Adicionalmente, una *cache* almacena durante unos minutos las rutas que se han ido usando (se muestra con `ip r l cache`).

Observar que en los comandos usamos dos abreviaturas distintas: `ru` para las reglas de la política de rutado y `r` para las tablas de rutas. Veamos dos ejemplos.

9.2.1. Ejemplo sencillo de rutado según origen

Imaginemos un router conectado a un conjunto de hosts a través del interface *eth0* (192.168.0.2) y que los comunica con Internet por una ADSL rápida por su interface *eth1* (192.168.10.2) y por una la RDSI lenta por *eth2* (192.168.20.2).



Host1 (de dirección 192.168.0.35) no se ha portado bien así que lo vamos a redireccionar por la conexión lenta. Introducimos en el fichero de configuración una regla de número 200 (menor que 256) que apunte a la tabla de rutas "Host1":

```
>> echo 200 Host1 >> /etc/iproute2/rt_tables
>> ip ru a from 192.168.0.35 table Host1
>> ip ru l
0:      from all lookup local
32765:  from 192.168.0.35 lookup Host1
32766:  from all lookup main
32767:  from all lookup default
```

Esta regla especifica que todo el tráfico que viene con IP de origen 192.168.0.35 use la tabla de rutas llamada "Host1". Necesitamos entonces crear la tabla de rutas "Host1" indicando la pasarela a la que se conecta el RDSI, 192.168.20.3:

```
>> ip r a default via 192.168.20.3 dev eth2 table Host1
>> ip r flush cache
```

Es importante distinguir entre *reglas* (abreviatura `ru`) y *tablas de rutas* (abreviatura `r`). Así, varias reglas pueden apuntar a la misma tabla y una tabla de rutas puede no tener ninguna regla apuntando a ella sin dejar de existir.

9.2.2. Ejemplo más complejo con marcación de paquetes

En el ejemplo previo hemos dsicriminado los paquetes según su IP de origen. Si queremos, por otra parte, basar nuestra política sobre el tipo de tráfico (correo, web, vídeo), usaremos la herramienta de marcación de paquetes *iptables*.

Siguiendo con el ejemplo, veamos ahora como desviar nuestro tráfico SMTP saliente (que nos envían nuestros clientes desde la red *eth0*) hacia la RDSI lenta de interfaz *eth2* 192.168.20.2. Marcamos los paquetes entrantes al sistema (PREROUTING) y con destino el puerto 25 (SMTP) con un número "4":

```
>> iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 25 -j MARK --set-mark 4
```

Ahora dirigimos los paquetes marcados "4" a una tabla de rutas específica que, en este caso, llamamos "Correo":

```
>> echo 201 Correo >> /etc/iproute2/rt_tables
>> ip ru a fwmark 4 table Correo
>> ip ru l
0:      from all lookup local
32764:  from all fwmark 4 lookup Correo
32765:  from 192.168.0.35 lookup Host1
32766:  from all lookup main
32767:  from all lookup default
```

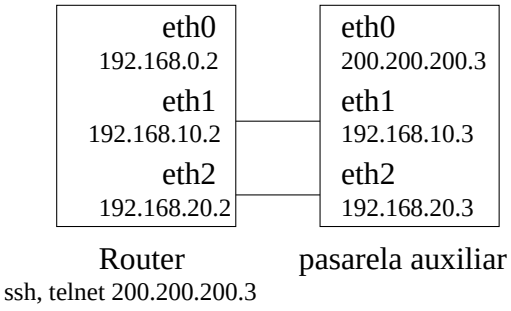
Sólo nos hace falta ahora crear la tabla de rutas "Correo":

```
>> ip r a default via 192.168.20.3 dev eth2 table Correo
```

Ya lo tenemos todo. Hemos usado una regla de marca muy simple, basada en el puerto destino del paquete. Con *iptables*, podemos hacerlo mucho más complejo, seleccionando usuarios, hora del día, contenidos, etc.

9.3. Ejercicios del comando `ip`

En estos ejercicios necesitaremos dos simulaciones Qemu que vamos a configurar paso a paso según el siguiente esquema:



1. Con el comando `ip`, asignar la dirección IP, máscara y dirección de broadcast a los interfaces de las dos máquinas (si el interface tenía previamente una dirección, borrarla primero). Después, activar los interfaces mediante la opción `up`. Testear que todo es correcto con `ip l l up` e `ip a l`.

2. Con el comando *ping* comprobar que se puede acceder al otro extremo de los enlaces, pero no a *200.200.200.3* desde el Router de la izquierda.

3. En el *Router*, con el comando `ip` asignar una ruta por defecto a través de la interface *eth1* en la tabla main y con la *gateway* correcta. Testear las rutas con *ip ru l* e *ip r l*.

4. Activar en la *Pasarela auxiliar* (Qemu de la derecha) la característica de pasar paquetes entre interfaces (*echo "1" > /proc/sys/net/ipv4/ip_forward*). Con el comando *ping* comprobar el acceso a *200.200.200.3* desde el *Router*.

⇒ ENTREGA 9. Rutado por usuario

El *Router* ya tiene asignada la ruta por defecto por el interface *eth1*. Ahora vamos realizar un rutado especial por el interface *eth2* para un usuario local [*usuario*] (elegir *debian*, *root* o crear uno nuevo).

1. Primero hay que marcar los paquetes generados localmente (*OUTPUT*) por el usuario [*usuario*] y cambiar la IP origen de los paquetes que salen por *eth2* (*POSTROUTING*) puesto que *iproute* hace rutado, pero no ajusta las direcciones IP de los paquetes que se han generado previamente. Esto se hace, respectivamente:

```
>> iptables -A OUTPUT -t mangle -m owner --uid-owner [usuario] -j MARK
   --set-mark [marca]
>> iptables -A POSTROUTING -o eth2 -t nat -j SNAT --to-source [IP_origen_eth2]
```

2. En segundo lugar, hay que crear una nueva tabla de rutas para los paquetes marcados con el número [*marca*] y rutarlos a través de *eth2*.

3. Puesto que en este ejemplo estamos usando una misma pasarela para las dos rutas (y no dos como ocurriría en el mundo real) debemos desactivar el filtro que verifica que la dirección de origen del paquete sea alcanzable desde el interface por el que llega.

```
echo "0" > /proc/sys/net/ipv4/conf/eth2/rp_filter
```

4. Para testear las rutas, en primer lugar, arrancar el capturador de paquetes (ethereal/wireshark) en el *Router*. En segundo lugar, iniciar para uno de los usuarios una conexión con *ssh 200.200.200.3* (no vale ping), salir y repetirlo para el otro usuario.

   Parar la captura y comparar con *ethereal* las rutas usadas para los dos usuarios examinado las direcciones IP y Ethernet de origen de los paquetes. En la lista tienen que aparecer algunos paquetes con direcciones 192.168.10.2 → 200.200.200.3 y otros con 192.168.20.2 → 200.200.200.3.

⇒ ENTREGA 10. Rutado por puerto

Rutar los paquetes generados localmente (OUTPUT) por diferente interface según su puerto de destino: 22 (ssh) por *eth1* y 23 (telnet) por *eth2*.

Podemos usar el esquema del ejercicio anterior y no es necesario borrar nada, basta con definir dos nuevas tablas y dos marcas (una para cada puerto) con iptables en la cadena OUTPUT. Si la práctica se hace de nuevo no olvidarse de ajustar las direcciones IP de los paquetes en el interface *eth2* y de desactivar el filtro que controla el interface de entrada.

Podemos testearlo con el capturador de paquetes comparando los paquetes de las transmisiones *ssh 200.200.200.3* y *telnet 200.200.200.3*.

⇒ ENTREGA 11. Balanceo del tráfico de salida entre dos interfaces

Sólo hay que poner la ruta por defecto de la tabla main con un balanceo del 50 % (pesos iguales) entre los dos interfaces *eth1* y *eth2*. Esto se hace con el siguiente comando:

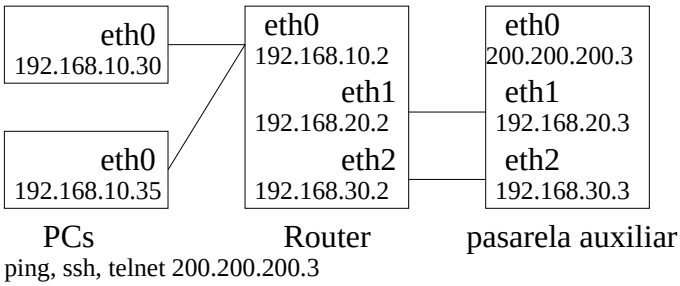
```
ip route add table main default scope global nexthop via [IP1] dev [IF1]
   weight 1 nexthop via [IP2] dev [IF2] weight 1
```

Para evitar problemas con los filtros los desactivamos en ambas máquinas:

```
echo "0" > /proc/sys/net/ipv4/conf/eth1/rp_filter
echo "0" > /proc/sys/net/ipv4/conf/eth2/rp_filter
```

Ojo, las trasmisiones con el mismo destino seguirán siempre la misma ruta, salvo que se borre la caché. Por tanto, para realizar el testeo podemos hacer sucesivas transmisiones por *ssh* (no vale *ping*) a 200.200.200.3 y borrar la caché entre transmisiones (`ip r l cache` e `ip r f cache`). Las transmisiones alternarán entre las dos rutas, lo cual puede comprobarse en la caché o en los paquetes capturados. También puede comprobarse que desactivando (down) uno de los interfaces los paquetes salen por el otro.

Ejercicio: Testeo del rutado según origen



En este ejercicio vamos a testear el funcionamiento del ruter según origen visto en los apartados 4.2.1 y 4.2.2 usando una pasarela y un PC auxiliar, de acuerdo con el esquema de la anterior figura. En este caso necesitamos tres Qemu (los dos PCs serán realizados con la misma simulación).

1. En el *Router* definimos los interfaces (las direcciones son diferentes a las de los ejercicios anteriores), definimos la ruta por defecto a través de *eth1* y activamos la característica *ip_forward*. A continuación, definimos una tabla de rutado *Host1* para el host origen 192.168.10.35 con ruta a través de *eth2*. Dada la configuración del Qemu, hay que desactivar el filtro que verifica que la dirección de origen del paquete sea alcanzable desde el interface por el que llega:

   ```
   echo "0" > /proc/sys/net/ipv4/conf/eth1/rp_filter
   ```

2. En la *Pasarela auxiliar* definimos los interfaces (las direcciones también son diferentes), definimos una ruta hacia 192.168.10.0/24 a través de *eth1* y activamos la característica *ip_forward*. En este caso debemos desactivar el siguiente filtro:

   ```
   echo "0" > /proc/sys/net/ipv4/conf/eth2/rp_filter
   ```

3. Para los dos *PCs* vamos a usar un único Qemu. Para simular el primero, asignamos la dirección 192.168.10.30 y la ruta por defecto a tráves de *eth0* y tecleamos *ping 200.200.200.3*. A continuación, pasamos a simular el segundo: cambiamos la dirección de *eth0* a 192.168.10.35, volvemos a poner la ruta por defecto a través de *eth0* (se ha borrado al cambiar la dirección) y tecleamos *ping 200.200.200.3*.

4. En el Router comprobamos mediante la cache por dónde han salido los paquetes. Tienen que aparecer las líneas 200.200.200.3 ← 192.168.20.2 y 200.200.200.3 → 192.168.30.2.

   ```
   ip r l cache | grep 200.200.200.3
   ```

5. Pasamos ahora a desviar el tráfico telnet (puerto 23) entrante en el Router por el interface *eth2*. En el router definimos una nueva tabla *Telnet* en función de la marca de los paquetes. Los paquetes telnet los marcamos con *iptables* en función del puerto 23. Limpiar la cache para no confundirse con las transmisiones anteriores.

6. En el PC iniciar primero una conexión *ssh* con 200.200.200.3 y luego una conexión *telnet*. Leer la cache en el router y comprobar que aparecen las dos rutas hacia 200.200.200.3.

10. Configuración de routers

NOTA: Usar dos simulaciones de Qemu con tres tarjetas de red sin configurar.

Existen varios programas que simulan los routers Cisco, entre los que están Quagga (`http://www.quagga.org/`) y GNS3 (Graphical Network Simulator-3) (`https://www.gns3.com/`). En esta práctica usaremos Quagga, aunque también se podría utilizar GNS3. Quagga dispone de una interfaz de configuración y comandos casi idénticos al Cisco IOS[14], e instala cinco procesos del sistema correspondientes a los protocolos de rutado, y que escuchan en puertos consecutivos:

zebra	2601 tcp
ripd	2602 tcp
ripngd	2603 tcp

ospfd	2604 tcp
bgpd	2605 tcp

10.1. Instalación

1. Lo primero es realizar la configuración básica en */etc/quagga*. En el fichero *daemons* indicamos que queremos *zebra*, *ripd*, *ospfd* y *bgpd* y editamos los correspondientes ficheros de configuración, *zebra.conf*, *ripd.conf*, *ospfd.conf* y *bgpd.conf*, etc. En */usr/share/doc/quagga/examples* tenemos ejemplos de la configuración. Por ejemplo, para zebra:

   ```
   ! Zebra configuration file
   hostname Router
   password zebra
   enable password zebra
   log file /tmp/fichero
   ```

   En donde damos un nombre al router, ponemos la clave de gestión, una segunda clave para los comandos privilegiados e indicamos dónde escribir los mensajes de auditoría. Otros comandos de configuración pueden consultarse en el manual. Editamos de igual forma los demás ficheros **.conf*.

   Arrancamos los procesos del sistema con `/etc/init.d/quagga restart`.

2. Pasamos a gestionar el router. Podemos acceder directamente a cada uno de los protocolos (daemons) que utiliza quagga. Simplemente hacemos un telnet al puerto que se desee, por ejemplo, para la configuración global: *telnet localhost zebra* y para la configuración de RIP: *telnet localhost ripd*.

   Ussndo el usuario *debian* con `telnet localhost zebra` obtenemos:

   ```
   Hello, this is Quagga (version 0.98.5).
   Copyright 1996-2005 Kunihiro Ishiguro
   Password:
   Router>
   ```

[14] Una guia sobre los comandos del Cisco IOS: `http://www.cisco.com/c/en/us/td/docs/ios/fundamentals/configuration/guide/15_1s/cf_15_1s_book.pdf`.

Una sesión típica toma la forma:

```
Router> ?
enable             Turn on privileged commands
exit               Exit current mode and down to previous mode
help               Description of the interactive help system
list               Print command list
show               Show running system information
who                Display who is on a vty
Router> enable
Password: XXXXX
Router# configure terminal
Router(config)# interface eth0
Router(config-if)# ip address 10.0.0.1/8
Router(config-if)# quit
Router(config)#
```

En esta sesión se observan los dos modos de operación: no privilegiado o de visualización (promt >) y privilegiado o de modificación (prompt #); al modo privilegiado se accede con el comando `enable`. La lista de comandos se obtiene con `?` (versión reducida) y con `list` (versión completa). El comando `help` da una ayuda general y `comando ?` explica algo el correspondiente comando.

Algunos comandos nos llevan a distintos submenús que se indican con distintos prompt, por ejemplo, Router(config)#, Router(config-if)#, etc. Cada submenú tiene su propia lista de comandos. Del submenú se sale con el comando `quit`.

3. *Comandos del modo terminal.* En el modo privilegiado, prompt Router#, algunos de los comandos son los siguientes:

   `write terminal`. Muestra la configuración actual.

   `write file`. Escribe la configuración actual al fichero de configuración (debe tener permiso de escritura para el usuario quagga tanto el fichero *zebra.conf* como el directorio de configuración).

   `show logging`. Muestra la configuración de auditoría.

   `who`. Muestra quién está conectado.

   `configure terminal`. Este comando es el primero que hay que teclear si se desea realizar alguna configuración.

4. *Comandos de interfaces de red.* Permiten asignar la dirección IP, máscara y demás características a cada interface de red. Desde el modo provilegiado, una vez hemos tecleado el comando `configure terminal` que nos lleva al menú de prompt Router(config)#, tecleamos el comando: `interface` *interface*, siendo el interface, *eth0*, *usb0*. Entramos en el menú de prompt Router(config-if)# con lo que dispondremos de los siguientes comandos:

`shutdown`. Desactiva el interface. (`no shutdown` lo activa).

Nota: casi todos los comandos admiten anteponerles la palabra `no`, lo que deshabilita los cambios hechos por el comando. Un ejemplo lo hemos visto en el comando anterior, y no lo repetiremos para el resto de los comandos.

`ip address` *address/prefix*. Asigna la dirección IPv4/máscara al interface, donde el parámetro es de la forma 193.144.84.0/24.

`description` *description*. Asigna un texto descriptivo al interface.

`multicast`: activa esta característica.

`bandwidth` *<1-10000000>*. Fija el ancho de banda en kbps que se utilizará como medida de costo cuando OSPF calcule las rutas.

5. *Comandos de rutas estáticas*. Pemiten especificar rutas estáticas. Suponemos que estamos en el menú de *configure terminal* indicado por el prompt Router(config)#.

   `ip route` *network gateway*. Establece una ruta a una determinada red (que se puede especificar de dos formas: *a.b.c.d/m* o *a.b.c.d m1.m2.m3.m4*) a través de una pasarela (que podemos especificar mediante una dirección IP o un interface). Por ejemplo:

   ```
   ip route 10.0.0.0/24 10.0.0.2
   ip route 10.0.0.0 255.255.255.0 10.0.0.2
   ip route 10.0.0.0/32 eth0
   ```

   Nota: la ruta por defecto se especifica como 0.0.0.0/0 y se utiliza para destinos que no coinciden con ninguna entrada de la tabla de rutas.

   `ip route` *network gateway <1-255>*. Similar al comando anterior, pero permite indicar la distancia de la ruta medida en el número de routers.

   `ip forwarding`. Activa la característica */proc/sys/net/ipv4/ip_forward* del kernel, imprescindible para hacer la transmisión de paquetes entre los interfaces de un router Linux.

6. *Comandos de visualización*. Estos comandos se introducen desde el menú principal, prompt Router>:

   `show interface`. Muestra la lista y configuración de los interfaces de red. El comando `show interface` *interface* muestra la configuración de un determinado interface.

   `show ip route`. Muestra las rutas, indicando si es conexión directa (D), estática (S), obtenida por por el kernel (K), por el protocolo RIP (R), OSPF (O), BGP (B), etc. Se le puede añadir un parámetro para especificar que muestre sólo un determinado tipo de ruta (connected, static, rip, etc)

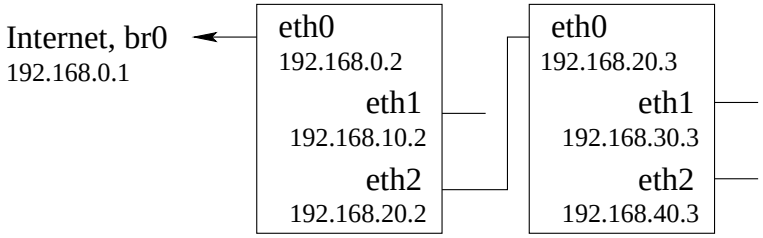
o para que muestre sólo la ruta que lleva a una determina red, por ejemplo, `show ip route` *network*.

`show ip forward`. Muestra si el kernel de Linux tiene activada la característica de transmisión de paquetes entre interfaces, imprescidible para funcionar como router (leer también el fichero */proc/sys/net/ipv4/ip_forward*).

`show memory`. Muestra la memoria ocupada por las distintas estructuras de los protocolos de rutado.

⇒ ENTREGA 12. Ejercicios de Zebra

En estos ejercicios necesitamos dos simulaciones Qemu con tres interfaces de red. La configuración que hagamos en este primer ejercicio será utilizada en los restantes (no olvidarse de grabarla en fichero).



1. Acceder a zebra con telnet y desde aquí grabar la configuración actual a fichero. Leer desde el S.O. lo que se ha grabado en *zebra.conf*.

2. Obtener con *show* la lista de interfaces de red del router. Asignar a cada uno de ellos una dirección IP y una mascara según el esquema de la siguiente figura y, si están desactivados, activarlos. Una vez configurados, volver a mostrarlos con *show*.

   Comprobar que desde el segundo router podemos hacer *ping* a 192.168.20.2 pero no a 192.168.10.2.

3. Activar desde zebra la opción del kernel *ip_forward*. Mostrar el estado de esta opción con *show*. Si Zebra no ha podido hacerlo, hacerlo desde el S.O.: `echo "1" > /proc/sys/net/ipv4/ip_forward`.

4. Definir desde zebra la ruta por defecto, para el primer router a través de *br0* y para el segundo a través del enlace con el primer router. Mostrar con *show* las rutas.

   Comprobar que desde el segundo router ya podemos hacer *ping* a 192.168.10.2.

5. Grabar la configuración actual al fichero de configuración *zebra.conf*.

6. Con *show* mostrar en pantalla la lista de comandos tecleados, la versión del programa y la configuración actual (buscar los comandos en la ayuda).

7. Desde zebra, mostrar en pantalla las opciones de auditoría activadas y desde Linux leer el fichero que nos indica con la auditoría recopilada.

10.2. Configuración de RIP (v1 y v2)

RIP es un sencillo protocolo de rutado de *Vector de Distancias* en el cual los mensajes se envían sólo a los routers vecinos. La versión 2 añade algunas funcionalidades como la autentificación. Las versiones 1 y 2 trabajan sólo con IPv4, pero existe una nueva versión, RIPng que trabaja con IPv6 (en quagga se trata como un protocolo distinto).

Para que funcione en quagga el protocolo RIP debemos tener activados por los menos los procesos *zebra* y *ripd*. Al menú de RIP accedemos con `telnet localhost ripd` (o con `vtysh`, ver el apartado final) y desde el submenú de *configure terminal*. Para activar y desactivar el protocolo RIP v1 y v2 tenemos los comandos:

```
router rip
no router rip
```

Una vez en el menú de RIP, de prompt Router(config-router)# disponemos de los comandos siguientes[15]:

1. *Comandos de configuración.*

   `version` *version*. Podemos establecer la versión del protocolo RIP, 1 o 2. El defecto es la versión 2.

   `network` *ifname*. Activa el protocolo RIP en el correspondiente interface.

   `network` *network*. Activa el protocolo RIP en el interface que tiene asignada la correspondiente red. Ejemplos se muestran un poco más abajo.

   `passive-interface` [*interface*|`default`]. Un interface es activo cuando envía mensajes RIP y pasivo cuando sólo los recibe. En el modo pasivo un interface no enviará paquetes RIP excepto a los routers vecinos indicados con el comando *neighbor*. Si se indica `default` todos los interfaces se harán pasivos.

   `neighbor` *a.b.c.d*. Activa el envío de mensajes RIP personalizados para los routers vecinos especificados. Puede utilizarse para los routers que no comprenden el multicast.

   Estos comandos también pueden escribirse en el fichero de configuración *ripd.conf*, por ejemplo:

   ```
   ! rip configuration
   ```

[15] *Nota:* casi todos los comandos se desactivan si se les antepone la palabra `no`, por lo que no lo vamos indicar individualmente en cada comando.

```
router rip
  network 10.0.0.0/8
  network eth0
```

2. *Comandos RIP de un interface*. Dentro del menú de cada uno de los interfaces (ver un apartado anterior), de prompt Router(config-if)#, se pueden configurar algunas cuestiones relacionadas con RIP. En concreto:

   `ip rip send version` *version*; e `ip rip receive version` *version* permiten especificar para cada uno de los interfaces la versión de RIP, 1 o 2, con la que se enviarán o recibiran paquetes.

   `ip split-horizon`. Esta propiedad reduce la probabilidad de lazos en las rutas.

3. *Anuncio de rutas*. Permite que las rutas obtenidas por otros procedimientos se incorporen a las tablas RIP. Los comandos se introducen desde el menú de RIP, prompt Router(config-router)#.

   `route` *a.b.c.d/m*. Asigna una ruta estática a la tablas RIP, aunque es preferible definir la ruta estática en zebra y redistribuirla.

   `redistribute static`. Redistribuye la información sobre las rutas estáticas a las tablas RIP.

   `redistribute ospf`. Redistribuye la información que posee el protocolo OSPF a las tablas RIP.

   `redistribute kernel`. Redistribuye la información de rutado que posee el kernel a las tablas RIP.

4. *Filtros en las tablas RIP*. La tabla RIP de un router pueden filtrase usando una lista de redistribución, es decir, que se puede especificar qué rutas se van a aceptar (entrada) o a hacer públicas (salida) y cuáles no.

   `distribute-list` *access_list* [`in|out`] *ifname*. Con este comando asignamos al interface *ifname* la lista de acceso denominada *access_list* para entrada o salida de los mensajes RIP. La lista de acceso se construye con las sentencias siguientes:

   `access-list` *access-list* [`deny|permit`] `any`

   `access-list` *access-list* [`deny|permit`] *a.b.c.d/m*

   En el siguiente ejemplo, *eth0* aceptará rutas sólo a la red especificada e ignorará todas las demás:

```
router rip
  distribute-list LISTA in eth0
access-list LISTA permit 10.0.0.0/8
access-list LISTA deny any
```

5. *Distancia y temporizadores.*

   `distance` <*1-255*>. Asigna el alcance de las rutas RIP, por defecto es 120.

   `timers basic` *update timeout garbage*. Asigna valores a los temporizadores, ver el manual.

6. *Autentificación.* RIPv2 permite dos tipos de autentificación: *text* y *MD5*. Si se activa la autentificación, los interfaces que se transmiten mensajes RIP deberán tener la misma clave. Dentro del menú de configuración de cada uno de los interfaces, de prompt Router(config-if)#, tenemos los comandos

   `ip rip authentication mode text`. Fija modo de autentificación por clave.

   `ip rip authentication string` *string*. Establece la clave *string*.

   `ip rip authentication mode md5`: Fija modo de autentificación MD5.

   `ip rip authentication key-chain` *key-chain*. Establece la clave *key-chain*, previamente definida. Por ejemplo:

```
key chain test
  key 1
    key-string test
! quit, quit
interface eth1
  ip rip authentication mode md5
  ip rip authentication key-chain test
```

7. *Comandos de depuración.* Los siguientes comandos activan la auditoría de los correspondientes eventos. La información se muestra o almacena en la forma indicada en la configuración por la sentencia `log` (en consola, en fichero, etc). Se introducen desde el menú de configuración del terminal.

   `debug rip events`. Muestra los paquetes enviados y recibidos, los temporizadores, y los cambios en los interfaces debidos a RIP.

   `debug rip packet`. Muestra información detallada de los paquetes RIP.

   `debug rip zebra`. Muestra la interacción entre zebra y RIP.

   `show debugging rip`. Indica que comandos de depuración están activados.

8. *Comandos de visualización.* Muestra en pantalla la información solicitada. Estos comandos se pueden introducir desde el modo no privilegiado, prompt Router>. Algunos de ellos son:

   `show ip rip`. Muestra en pantalla las rutas RIP.

   `show ip rip status`. Muestra en pantalla el estado de RIP, incluyendo la versión, interfaces y filtros.

⇒ ENTREGA 13. Ejercicios de RIPv2

1. Antes de comenzar con RIP, mostrar con los comandos de visualización los interfaces y las rutas y asegurarse que los dos routers estén configurados como en la práctica de Zebra (lo estarán si hemos grabado la configuración a fichero). A continuación, arrancar el capturador de paquetes *ethereal*.

2. Acceder a RIP usando telnet y activar la auditoría de todo lo relacionado con el protocolo RIP. Activar el protocolo RIP en todos los interfaces.

3. Comprobar si se añadido alguna entrada nueva en la tabla de rutas. Leer el fichero de auditoría. Observar si *ethereal* ha recibido algún paquete RIP.

4. Identificar varios paquetes RIP con *ethereal* y deducir las entradas de la tabla de rutas a las que corresponderían.

5. Comprobar si el destino de los paquetes RIP es una dirección multicast (se reconoce al ser 224.0.0.x). Comprobar la dirección con *ping 224.0.0.x*.

6. Definir algún router vecino (dentro de la red asignada a un interface) como neighbor y comprobar con *ethereal* que se le envían mensajes personalizados. Desactivar esta característica.

7. Mostrar en pantalla toda la información de Quagga relacionada con RIP.

8. Identificar la rutas que hay almacenadas en las tablas y obtener cuáles son estáticas, directamente conectadas, obtenidas por el kernel, por RIP, etc.

9. Mostrar en pantalla la lista de comandos tecleados y la configuración actual. Escribir la configuración actual a fichero.

10. Por último, desde Linux echar un vistazo al fichero de auditoría.

10.3. Configuración de OSPF

OSPF es un protocolo de rutado más avanzado que RIP. Es un rutado de tipo de *Estado de Enlaces* en el que los mensajes se envían a routers no forzosamente vecinos. También es un rutado jerárquico que divide la red en áreas.

Un router OSPF debe mantener una base de datos con la topología de la red a la que pertenece. La base de datos topológica contiene una colección de LSAs (Link-State Advertisements) recibidos desde el resto de los routers del área. Un router, después de asegurarse que sus interfaces funcionan, envía mensajes Hello para ponerse en contacto con sus vecinos (neighbor). Estos mensajes también le permiten a un router saber si sus vecinos son todavía funcionales.

Para que funcione en quagga el protocolo OSPF debemos tener activados por los menos los procesos *zebra* y *ospfd*. Al menú de OSPF accedemos con `telnet`

`localhost ospfd` (o con `vtysh`) y desde el submenú de *configure terminal*. Los comandos para activar o desactivar el protocolo OSPF son:

```
router ospf
no router ospf
```

Una vez en el submenú OSPF de prompt Router(config-router)# disponemos de los comandos siguientes[16]:

1. *Comandos de configuración.*

   `ospf rfc1583compatibility`. Fija el comportamiento del protocolo de rutado de forma que la selección de las rutas se haga de acuerdo con los estándares rfc1583 o rfc2328.

   `ospf abr-type` *type*. Fija el comportamiento de los routers de frontera de área que según el campo *type* puede ser: cisco, ibm, shortcut, standard.

   `ospf router-id` *a.b.c.d*. Pone el identificador del router.

   `passive interface` *interface*. Suprime el envío de los mensajes HELLO del correspondiente interface.

2. *Comandos de área*. Una red OSPF puede dividirse en varias partes denominadas áreas. Un área es una colección lógica de redes, routers y enlaces que tienen el mismo *area-id*. La división en áreas debe verificar lo siguiente:

   (1) Dede existir un área troncal que interconecte las diferentes áreas.

   (2) Cada área no trocal debe estar conectada al área troncal, aunque la conexión sea lógica a través de un enlace virtual.

   (3) El área troncal no puede subdividirse en subáreas.

   La áreas se identifican por un *area-id* que puede especificarse mediante un entero o una dirección: 0-4294967295 o A.B.C.D. Por ejemplo, 0.0.0.0 es igual al área 0. Si hay varias áreas en una red, la red trocal debe tener el *area-id* 0. Si el router tiene dos interfaces (o más) en diferentes áreas, entonces es un router de frontera de área (ABR, Area Border Router).

   `network` *a.b.c.d/m* `area` *area-id*. Asigna una red al protocolo OSPF y a una determinada área.

   `area` *area-id* `range` *a.b.c.d/m*. Crea un rango de direcciones para las rutas internas a un área las cuales serán anunciadas a otros áreas por un ABR. Por ejemplo:

   ```
   router ospf
     network 192.168.1.0/24 area 0.0.0.0
     network 10.0.0.0/8 area 0.0.0.1
     area 0.0.0.1 range 10.0.0.0/8
   ```

[16] *Nota:* casi todos los comandos se desactivan si se les antepone la palabra `no`, por lo que no lo vamos indicar individualmente en cada comando.

`area` *area-id* `range` *a.b.c.d/m* [`advertise|not-advertise`]. Especifica si el rango de direcciones creado será anunciado a otras áreas por un ABR.

`area` *area-id* `stub`. Define un área como un *stub*. Las areas *stub* no son incluidas en rutas externas ni reciben anuncios desde el exterior.

`area` *area-id* `default-cost` [*cost*|`infinity`]. Define el costo de la ruta por defecto en un área *stub*. Por ejemplo:

```
router ospf
  network 10.1.1.0/24 area 0
  network 192.12.12.0/24 area 1.1.1.1
  area 1.1.1.1 stub
  area 1.1.1.1 default-cost 20
```

`area` *area-id* `virtual-link` *A.B.C.D*. Crea un enlace lógico desde el área de *area-id* hasta el router especificado por la dirección IP *A.B.C.D*.

`area` *area-id* `export-list` *NAME*. Expecifica a qué otras áreas son anunciadas las rutas internas de un determinado área. Por ejemplo:

```
router ospf
  network 192.168.1.0/24 area 0.0.0.0
  network 10.0.0.0/8 area 0.0.0.10
  area 0.0.0.10 export-list LISTA
! quit
access-list LISTA permit 10.10.0.0/16
access-list LISTA deny any
```

En este ejemplo, las rutas internas del área 0.0.0.10 y con el rango 10.10.0.0/16 son anunciadas a la red 10.10.0.0/16, pero no a las demás.

`area` *area-id* `authentication`. Activa la autentificación. La clave ha de ser la misma en los interfaces que comunican entre sí. Por ejemplo:

```
router ospf
  network 10.0.0.0/24 area 0.0.0.0
  area 0.0.0.0 authentication
! quit
interface eth0
  ip ospf authentication-key adcdefgh
```

3. *Comandos OSPF de un interface*. Estos comandos se introducen en el menú de configuración de cada interface (no en el menú del router ospf). Una vez dentro de *configure terminal*, tecleamos `interface` *interface* y dispondremos de los siguientes comandos:

   `ip ospf authentication-key` *key*. Asigna la clave para la autentificación por texto.

`ip ospf message-digest-key` *<1-255>* `md5` *key*. Asigna la clave para la autentificación MD5.

`ip ospf cost` *<1-65535>*. Asigna el costo para el enlace del interface.

`ip ospf hello-interval` *<1-65535>*. Asigna el tiempo entre envíos del mensaje HELLO. El valor por defecto es 10 s.

`ip ospf dead-interval` *<1-65535>*. Asigna el tiempo de espera en segundos antes de que un router declare muerto a un vecino. Defecto: 40 s.

`ip ospf retransmit-interval`. Asigna el valor del temporizador por el cual se reenvían los mensajes LSA (anuncios de estado de enlaces) si no se reciben antes los reconocimientos.

`ip ospf network` [`broadcast|non-broadcast|point-to-multipoint|point-to-point`]. Especifica el tipo de enlace del interface.

`ip ospf priority` *<0-255>*. Coloca la prioridad del router en redes miltiacceso, es decir en aquellas redes a las que se accede a través de más de un router. El valor por defecto es de 1.

4. *Comandos de redistribución de rutas en OSPF.*

   `redistribute` [`kernel|connected|static|rip|bgp`]. Redistribuye la información de rutas desde otros protocolos de rutado a OSPF.

5. *Comandos de depuración.* Activan la realización de auditorías. La información solicitada se muestra en la forma indicada en la configuración mediante la sentencia `log` (en consola, en fichero, etc). Se introducen desde el menú de configuración del terminal.

   `debug ospf event`. Habilita la auditoría de los eventos OSPF.

   `debug ospf packet` [`hello|dd|ls-request|ls-update|ls-ack|all`]. Habilita la auditoría de los paquetes transmitidos del tipo especificado.

   `debug ospf lsa`. Habilita la auditoría de LSA (link-state advertisements).

6. *Comandos de visualización.* Saliendo del menú principal, prompt Router>, tenemos los comandos:

   `show ip ospf`. Muestra en pantalla la información global de OSPF.

   `show ip ospf interface` *interface*. Muestra la información OSPF referida a un determinado interface.

   `show ip ospf neighbor`. Muestra información de los routers vecinos.

   `show ip ospf database`. Muestra Información sobre la tabla de LSAs (link-state advertisements).

   `show ip ospf route`. Muestra información sobre las rutas OSPF.

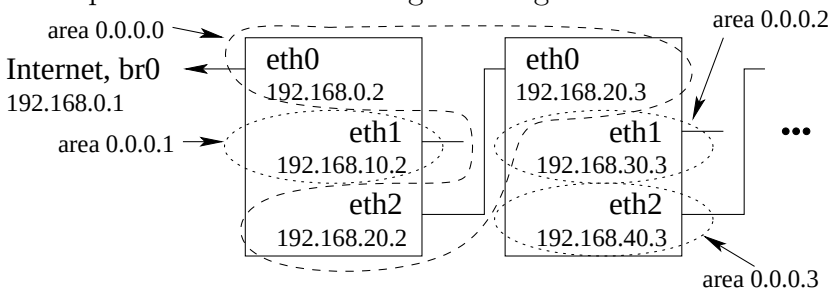
⇒ ENTREGA 14. Ejercicios de OSPF

1. En primer lugar, escribir el fichero de configuración de OSPF y desactivar en *daemons* la línea RIP para que este protocolo no interfiera. Al reiniciar el servicio nos va a decir que hay un problema de permisos, por lo que deberemos asignar los ficheros de configuración al usuario y grupo *quagga*.

2. Visualizar las rutas iniciales (esperar a que se borren las obtenidas por RIP). Arrancar el capturador de paquetes *ethereal*.

3. Acceder a OSPF usando telnet y activar la auditoría de todo lo relacionado con el protocolo OSPF.

4. Activar el protocolo OSPF en todos los interfaces de dos routers definiendo las áreas que se muestran en la siguiente figura:



5. Comprobar si se han creado nuevas rutas. Leer el fichero de auditoría. Observar en *ethereal* si se ha transmitido algún paquete OSPF.

6. Identificar algún paquete HELLO y alguno LSA. En estos últimos, encontrar algún anuncio de ruta y deducir cuál es la ruta.

7. Mostrar las rutas.

8. Mostrar en pantalla la lista de comandos tecleados y la configuración actual. Escribir la configuración actual a fichero.

9. Por último, desde Linux echar un vistazo al fichero de auditoría.

10.4. Configuración de BGP

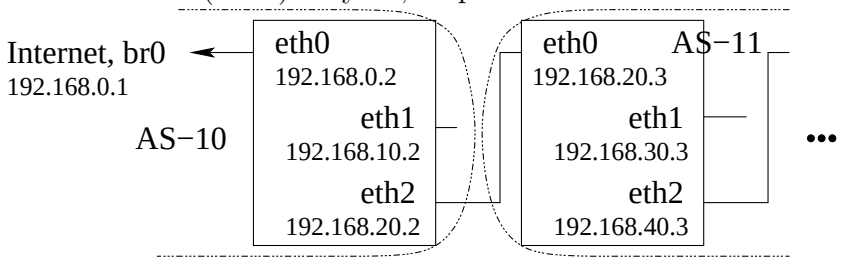
BGP es un protocolo de rutado entre sistemas autónomos, al contrario que RIP y OSPF, que rutan dentro del mismo sistema autónomo. Algunos comandos de BGP se verán en los ejercicios.

⇒ ENTREGA 15. Ejercicios de BGP

1. En primer lugar, crear el fichero de configuracion de BGP, desactivar en *daemons* el protocolo OSPF para que no interfiera, reiniciar el servicio y esperar a que se borren las rutas obtenidas por OSPF. Arrancar el capturador de paquetes *ethereal*.

2. Acceder a BGP usando telnet, consultar en la ayuda la lista de comandos y activar la auditoría de todo lo relacionado con el protocolo BGP.

3. Activar el protocolo BGP en los dos routers indicando los números de sistemas autónomos (ASN) 10 y 11, respectivamente.



4. Definir para cada router su vecino BGP. Por ejemplo, para el router de la izquierda: `neighbor 192.168.20.3 remote-as 11`. Asignar el protocolo BGP en todos los interfaces, por ejemplo, `network 192.168.0.0/24`, etc.

5. Comprobar si se han creado nuevas rutas. Leer el fichero de auditoría. Observar en *ethereal* si se ha transmitido algún paquete BGP.

6. Identificar algunos paquetes OPEN, KEEPALIVE y UPDATE. En estos últimos, encontrar algún anuncio de ruta y deducir cuál es la ruta.

10.5. El Shell VTY

Quagga proporciona la herramienta `vtysh` que integra todos los protocolos (RIP, OSPF, ...) y que permite realizar la configuración, gestión y auditoría de un modo centralizado, además de incorporar algunos comandos que normalmente se invocan desde Linux, como `ping` o `traceroute`. La herramienta se denomina VTY (Virtual TeleType) y puede accederse con el comando `vtysh`.

10.5.1. Ejercicios del Shell VTY

1. Editar el fichero de configuración *vtysh.conf* e incluir dos líneas: `hostname Vtysh` y `username debian nopassword`. Entrar en la herramienta tecleando `vtysh` (aparece una pantalla en blanco, a la que respondemos `q`). Probar algún comando y entrar en la configuración de RIP, OSPF, etc.

11. Dispositivos de red virtuales y túneles

11.1. Interfaces virtuales

Los interfaces *tun/tap* son dispositivos de red de tipo software, es decir que sólo existen en el kernel de Linux y no tienen ningún componente hardware físico asociado (tarjetas de red). Cuando un programa se conecta a un interface tun/tap, obtiene un descriptor de archivo especial. La lectura le proporciona al programa los datos que el interface está enviando, mientras que los datos que el programa escriba (con el formato adecuado) aparecerán como entrada en el interface. La diferencia entre ambos tipos es que la salida de un interface *tun* es un paquete IP sin cabecera Ethernet, mientras que en el caso de un interface *tap* es una trama Ethernet completa. Un tutorial de programación está disponible en la página *backreference.org*[17].

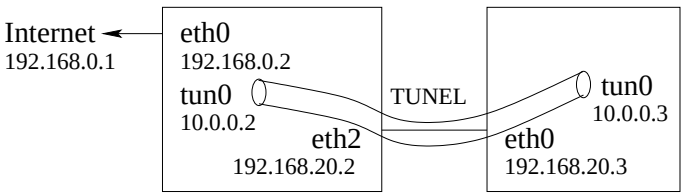
El comando *tunctl* permite crear interfaces tun/tap. Los formatos para crear o borrar un interface que pasará a ser propiedad de un usuario son:

`tunctl -u` *user* `-t` *tap0*
`tunctl -d` *tap0*

Los interfaces virtuales se usan en emuladores y para construir túneles.

11.2. Túneles

Existe multitud de programas para hacer túneles y en este apartado probaremos tres de ellos. Muchos admiten cifrado, IPV6, están pensados para construir redes privadas virtuales (VPN), etc. Se trata realmente de encapsular los paquetes del tipo deseado dentro de los paquetes IP estándar. El esquema típico de un túnel se muestra en la siguiente figura:



El correspondiente programa de túnel crea un interface virtual en cada uno de los extremos, para lo cual hay que indicarle las direcciones de los interfaces reales (en la figura 192.168.20.2 y 192.168.20.3). Esto usualmente hay que hacerlo independientemente para cada uno de los extremos.

A continuación hay que configurar las direcciones IP de los interfaces virtuales (en la figura 10.0.0.2 y 10.0.0.3) y las rutas para esos interfaces de ls forma usual con lo comandos *ifconfig* y *route*. Por ejemplo, para el extremo de la derecha, suponiendo que queremos que la ruta por defecto sea a través del túnel:

[17] `http://backreference.org/2010/03/26/tuntap-interface-tutorial/`

```
>> ifconfig tun0 10.0.0.3 pointopoint 10.0.0.2
>> route add -net 10.0.0.0/8 dev tun0
>> route add default gw 10.0.0.2 dev tun0
```

11.2.1. Comando ip

Sin necesidad de usar herramientas externas podemos crear túneles sin cifrar, cifrados o IPv6, con el propio comando `ip`, que forma parte del paquete *iproute2* de Linux. Existen tres tipos de túneles disponibles:

Tipo	Característica	Módulo necesario
ipip	IP sobre IP	
gre	IP cifrado sobre IP	`modprobe ip_gre`
sit	IPv6 sobre IPv4	`modprobe ipv6`

Para los dos últimos tipos antes de crear el túnel hay que cargar los módulos correspondientes. El formato del comando para gestionar (crear, modificar, borrar, etc) un túnel es:

`ip tunnel [add|change|delete|show]` *nombre* `mode` *tipo* `local` *ip* `remote` *ip*

11.2.2. OpenVPN

Como su nombre indica, OpenVPN[18] es una aplicación multiplataforma adecuada para construir una Red Privada Virtual (VPN). Es decir, es un programa que configura túneles privados sobre la infraestructura pública de Internet. Es una aplicación muy completa que incluye diferentes tipos de cifrado simétrico y asimétrico. Si bien, las opciones se suelen poner en un fichero de configuración, también las podemos escribir en un comando. Por ejemplo, para crear un túnel cifrado con clave estática:

`openvpn --genkey --secret` *key.txt*
// copiar el fichero *key.txt* en el otro extremo, por ejemplo, con scp
`openvpn --dev` *nombre* `--local` *ip* `--remote` *ip* `--secret` *key.txt*

Una vez configurados los extremos el comando nos dice que la secuencia de inicialización se ha completado.

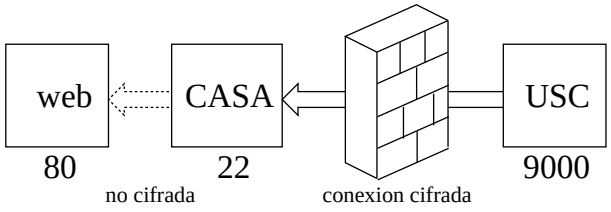
11.2.3. SSH

SSH es un protocolo y una aplicación para crear conexiones cifradas por las que se pueden realizar conexiones remotas, ejecutar comandos, transferir ficheros, redirigir el entorno gráfico etc. Los túneles creados por ssh pueden contener

[18] Disponible en `http://openvpn.net`. En la imagen de disco duro para Qemu está ya instalado.

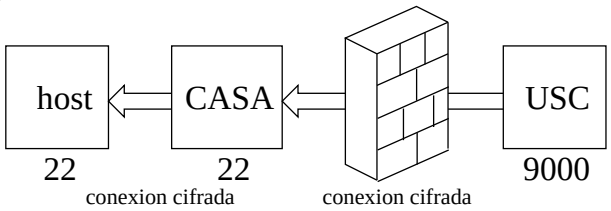
cualquier tipo de conexiones (ftp, telnet, http, etc), con la ventaja añadida del cifrado. SSH no usa interfaces virtuales, sino que sigue una arquitectura cliente-servidor. Los túneles ssh pueden crearse de varias formas usando tres tipos de mecanismos de redirección[19]:

1. *Local port forwarding.* En este ejemplo suponemos que debido al firewall de la USC no podemos conectarnos directamente a un servidor web externo, por lo que generamos una redirección local al puerto 9000 para conectarnos primero al ordenador de casa y desde allí al servidor web.



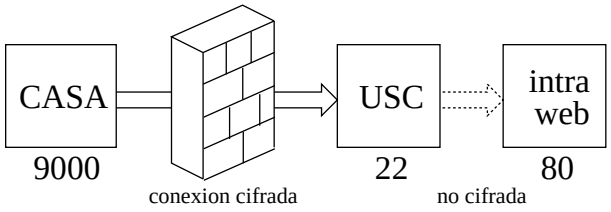

```
USC>> ssh -L 9000:web:80 casa
USC>> http://localhost:9000
```

   De forma similar, si desde la USC no podemos conectarnos directamente por ssh a un host, configuramos primero un túnel al ordenador de casa y luego un segundo túnel al host.




```
USC>> ssh -L 9000:localhost:9000 casa
USC>> ssh -L 9000:host:80 casa
USC>> ssh -p 9000 localhost
```

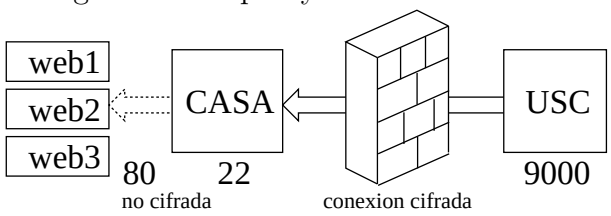
2. *Reverse port forwarding.* En este segundo ejemplo suponemos que desde casa podemos conectarnos a una de las máquinas de la USC, pero no podemos hacerlo a la web interna. Desde la máquina de la USC a la que tenemos acceso hacemos una redirección inversa de puertos:



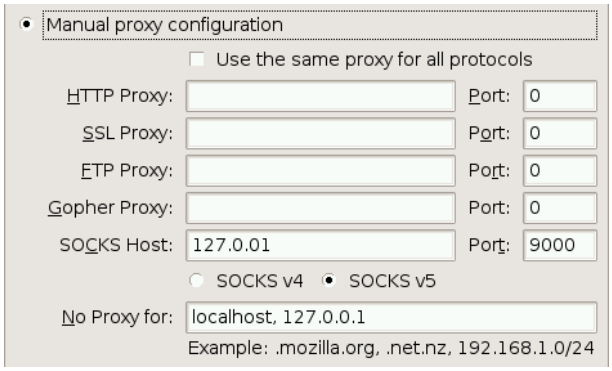
[19] `http://chamibuddhika.wordpress.com/2012/03/21/ssh-tunnelling-explained/`

```
USC>> ssh -R 9000:intra_web:80 casa
casa>> http://localhost:9000
```

3. *Dynamic port forwarding.* La redirección de puertos dinámica permite realizar conexiones remotas a todas las localizaciones. Sin embargo, la aplicación cliente que utiliza el túnel y se conecta al puerto local debe poder realizar la transmisión usando el protocolo SOCKS. En el caso de un navegador, esto se hace configurando un proxy.




```
USC>> ssh -D 9000 casa
USC>> http://web (a través del proxy)
```



⇒ ENTREGA 16. Ejercicio de túneles

(Para nota) En esta práctica configuraremos un túnel de cada uno de los tipos que hemos visto previamente.

1. Configurar los dos Qemu como se muestra en la primera figura del apartado. Para el primer Qemu, interfaces *eth0* 192.168.0.2 y *eth2* 192.168.20.2, ruta por defecto a través de la pasarela 192.168.0.1, retransmisión de paquetes (ip_forward) y NAT (iptables). Para el segundo Qemu configuramos *eth0* 192.168.20.3 (la ruta por defecto la pondremos después a través del túnel).

2. Crear un túnel con el comando ip entre los interfaces de direcciones 192.168.20.2 y 192.168.20.3 y de nombre *tun0* (tal como se muestra en la primera de las figuras del capítulo).

Asignarle a los interfaces virtuales las direcciones 10.0.0.2 y 10.0.0.3 y añadir a la tabla de rutas la red 10.0.0.0/8 a través del dispositivo *tun0*. Comprobar con el comando ping que tenemos respuesta del otro lado del túnel.

3. En el segundo Qemu, cambiar la ruta por defecto a través de la pasarela 192.168.20.2 por una a través del túnel. Comprobar que obtenemos respuesta al comando *host www.usc.es*.

4. Crear un segundo túnel con el comando openVPN entre los interfaces de direcciones 192.168.20.2 y 192.168.20.3 y de nombre *tun1* (de forma similar al caso anterior). Previamente habrá que generar un fichero de clave y transferirlo al otro extremo.

   Asignarle a los interfaces virtuales las direcciones 11.0.0.2 y 11.0.0.3 y añadir a la tabla de rutas la red 11.0.0.0/8 a través del dispositivo tun1. Comprobar con el comando ping que tenemos respuesta del otro lado del túnel.

5. Configurar una redirección de puertos dinámica con el comando ssh en el segundo Qemu. Usar como "casa" el interface de dirección 192.168.0.1 del PC real y un nombre de usuario válido para que nos deje hacer la conexión (esto es, escribir `"usuario@rai.usc.es"@192.168.0.1`). Podemos usar la opción `-v` para que nos dé información adicional (múltiples opciones `-v` aumentan la información, siendo el máximo de 3).

   En el segundo Qemu configurar en las preferencias del firefox, apartado de *network*, el proxy. Comprobar que con esta configuración podemos conectarnos a los servidores web de la USC y comprobar con ethereal que en los paquetes aparece el puerto 9000.

11.3. IPsec

IPsec es un conjunto de protocolos para añadir seguridad a las comunicaciones IP incorporando autentificación y cifrado. Es un estándar ampliamente utilizado en el campo empresarial para construir VPNs. Opera en la capa de red, y comprende dos elementos.

1. Cabeceras autentificadas (AH, Authentication Header), que garantiza la identidad e integridad del paquete, protegiendo la cabecera. Se suelen utilizar los algoritmos MD5 o SHA-1.

2. Cifrado de los datos (ESP, Encapsulating Security Payload), que garantiza la identidad, integridad y confidencialidad de los datos que incluye el paquete. Es posible utilizar DES, triple-DES, AES o Blowfish.

La configuración de la aplicación es sencilla, pues consiste básicamente en un script que indica los algoritmos y las claves para cifrar cabeceras y datos en cada

uno de los sentidos de la transmisión y las políticas de seguridad. Existen dos posibilidades para hacer esto, de forma manual con el comando *setkey* y de forma automática usando *racoon*.

Ejercicio de IPsec

En este ejercicio vamos a configurar de forma manual un túnel IPsec entre los interfaces *eth0* de dos máquinas virtuales.

1. Configurar el interface *eth0* del primer Qemu a 192.168.0.2 y para el segundo Qemu 168.168.0.3 y la ruta por defecto a través de la pasarela 192.168.0.1 en ambos. Nos traemos los paquetes que necesitamos, ya que por defecto no están instalados en la máquina virtual proporcionada:

   `export http_proxy=http://proxy2.usc.es:8080` (solo PCs laboratorio)

   `apt-get install ipsec-tools racoon`

   Cuando nos pregunte el modo de configuración para el proceso racoon IKE seleccionamos *racoon-tool*.

2. A continuación, escribimos los scripts *setkey.sh* en ambas máquinas. Para la primera de las máquinas,

```
#!/usr/sbin/setkey -f

flush;
spdflush;

add 192.168.0.2 192.168.0.3 ah 0x201 -A hmac-md5 CLAVE1;
add 192.168.0.3 192.168.0.2 ah 0x202 -A hmac-md5 CLAVE1;

add 192.168.0.2 192.168.0.3 esp 0x301 -E 3des-cbc CLAVE2;
add 192.168.0.3 192.168.0.2 esp 0x302 -E 3des-cbc CLAVE2;

spdadd 192.168.0.2 192.168.0.3 any -P out ipsec esp/transport//require ah/transport//require;
spdadd 192.168.0.3 192.168.0.2 any -P in ipsec esp/transport//require ah/transport//require;
```

   Donde CLAVE1 y CLAVE2 son cadenas de 16 y 24 bytes, para la protección de la cabecera y los datos del paquete, respectivamente. Estas claves deben generarse aleatoriamente para mayor seguridad, por ejemplo,

   `dd if=/dev/random count=16 bs=1 | xxd -ps`

   Estas dos claves serán las mismas en las dos máquinas. En el script de la segunda máquina, intercambiamos en las dos últimas líneas *in* y *out*.

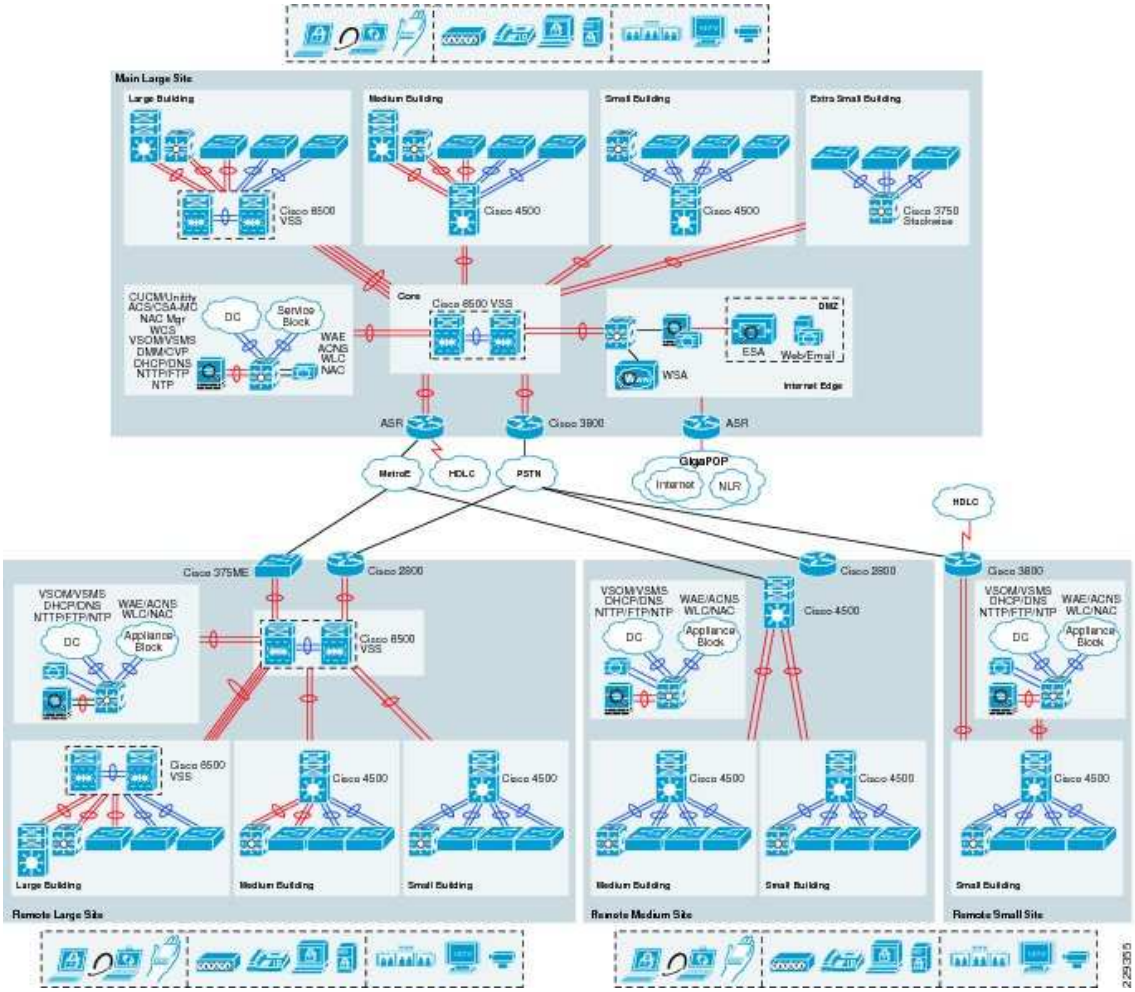
3. Damos permiso de ejecución a los scripts y ejecutamos en ambas máquinas.

4. Ponemos a funcionar el capturador de paquetes y cuando hagamos una conexión de una máquina a la otra, por ejemplo con ssh, veremos que el protocolo que aparece en los paquetes capturados es ESP.

12. Diseño de la red de una empresa

En el documento de la empresa CISCO *Medium Enterprise Design Profile Reference Guide* se presenta una guía para realizar el diseño de la red de una empresa de mediano tamaño. La guía está disponible en:

`http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Medium_Enterprise_Design_Profile/MEDP.html`



Realizar como ejercicio un trabajo de unas 4 páginas (incluyendo dibujos, esquemas y tablas) de cualquier apartado de esa guía. Para los dispositivos de red utilizados, buscar e incluir en el informe sus características y precio.