

# Obligatorio 3

## Forwarding, Routing y Switching

**Redes de Computadoras 2012**

**INCO – Facultad de Ingeniería – Udelar**

**Octubre 2012**

Grupo 59

Álvaro Acuña – CI: 3826062-8

Gabriel Centurión – CI: 2793486-8

Germán Mamberto – CI: 3187102-8

Fernando Mangino – CI: 3621009-1

### Contenido

Objetivo .....	3
Herramientas .....	3
Laboratorio de Routing y Forwarding (rip).....	4
a) Instalación del laboratorio.....	5
b) Captura de paquetes en las interfaces de r1 .....	6
c) Análisis del archivo ripd.startup.r1.cap .....	7
d) Continuación del laboratorio .....	12
e) Conexión a zebra en R3 mediante telnet.....	15
f) ¿Por qué es necesario agregar una ruta estática a 100.1.0.0/16 en r5? .....	16
g) ¿Por qué se instala una ruta por defecto en r4? ¿Por qué se redistribuye en RIP? .....	16
h) Comando <i>ping 193.204.161.1</i> exitoso en routers r1...r4.....	17
Laboratorio de Routing y Forwarding (ospf).....	18
a) ¿Se puede hacer desde en línea (sin re-levantar el laboratorio)? .....	18
b) Continuación del laboratorio con OSPF.....	21
c) Captura de paquetes en las interfaces de r1 en /hosthome/ospfd.startup.r1.cap .....	21
d) Análisis del archivo ospfd.startup.r1.cap.....	21
e) Continuación del laboratorio .....	32
f) ¿Cómo se redistribuye la ruta por defecto hacia r5 en OSPF? .....	32
g) Conexión al demonio ospf en r3 e identificación de las rutas externas de OSPF tipo 2 (E2) .....	36
h) Conexión al demonio zebra en r3. ....	38
Laboratorio de Bridging (two-switches) .....	39
a) Instalación del laboratorio de Bridging .....	39
b) Vinculación de lab.conf con la topología de red.....	39
c) Verificación del plan de numeración (IP y MAC) con la diapositiva. ....	41
d) Modificación del plan de numeración IP para pertenecer al prefijo 172.31.0.0/24.....	41
e) Modificación de direcciones MAC para emular tarjetas de red de Speakercraft Inc. ....	41
f) Comandos utilizados para realizar las dos partes anteriores. ....	42

# Objetivo

Comprender los conceptos básicos de Routing, Forwarding y Switching a través de la realización de tres laboratorios con diferentes topologías.

Se observará, configurará y analizará el comportamiento de los diferentes escenarios, y evidenciarán los resultados en la documentación.

# Herramientas

El laboratorio se realizará en una máquina virtual que contiene una distribución de la herramienta de emulación Netkit, basada en una versión recortada de Knoppix 6, que además incluye, entre otras, la herramienta Wireshark (Netkit-2.7-K2.8-F5.1.iso).

Netkit permite emular redes compuestas por diversos dispositivos de red como ejemplo computadoras, routers y switches. Los dispositivos de red son emulados como máquinas virtuales uml (user-mode linux) basadas en Debian, y se pueden interconectar mediante enlaces ethernet emulados.

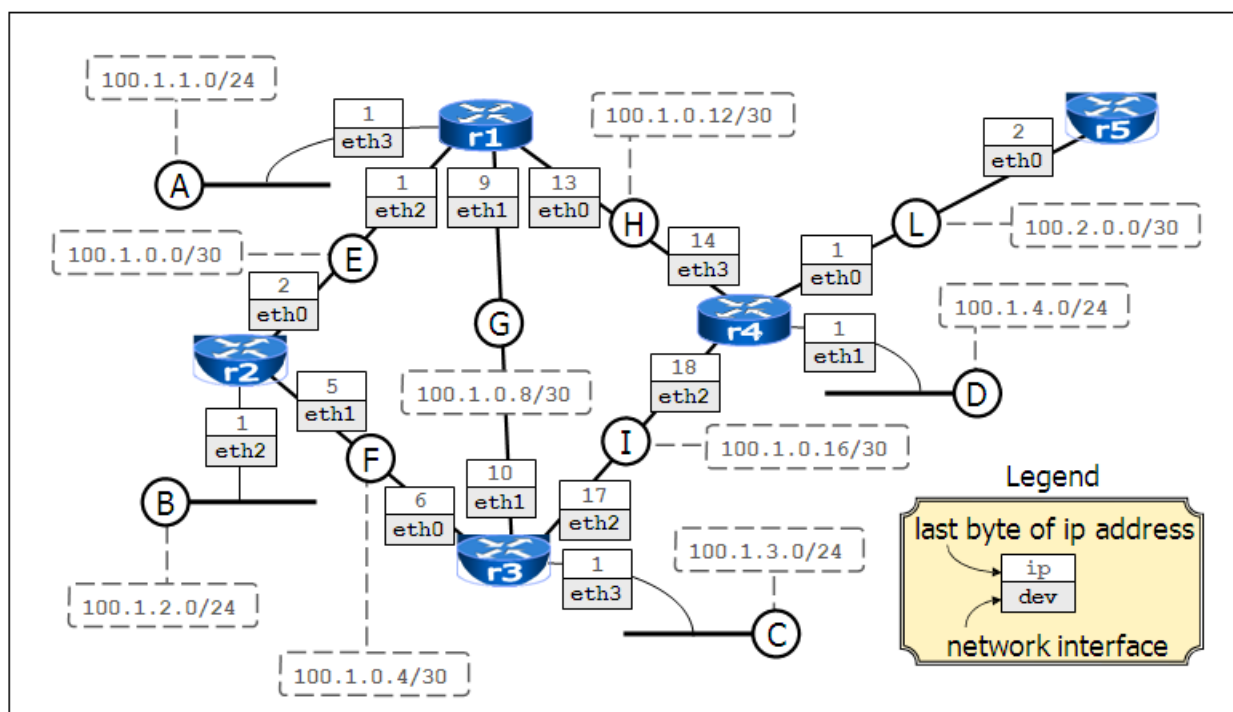
# Laboratorio de Routing y Forwarding (rip)

En esta primera parte del laboratorio estudiaremos el protocolo Routing Information Protocol en su segunda versión: RIPv2

Este es un protocolo de encaminamiento dinámico de tipo IGP (Internal Gateway Protocol), mediante el cual los routers pertenecientes a un mismo sistema autónomo intercambian y actualizan sus correspondientes tablas de rutas.

También se clasifica como un protocolo de enrutamiento del tipo vector de distancia (distance-vector) que utiliza el número de saltos como métrica hasta alcanzar la red de destino. Si existen dos rutas posibles para alcanzar un mismo destino, RIP elegirá la ruta que presente el menor número de saltos. El límite máximo de saltos en RIP es de 15, por lo que 16 se considera una ruta inalcanzable. Las actualizaciones de rutas se transmiten cada 30 segundos.

Al arrancar el siguiente laboratorio, con el comando `lstart`, se genera la siguiente topología de red en la cual intervienen 5 hosts virtuales (routers r1,r2,r3,r4,r5) y 10 redes (A,B,C,D,E,F,G,H,I,L):



### a) Instalación del laboratorio

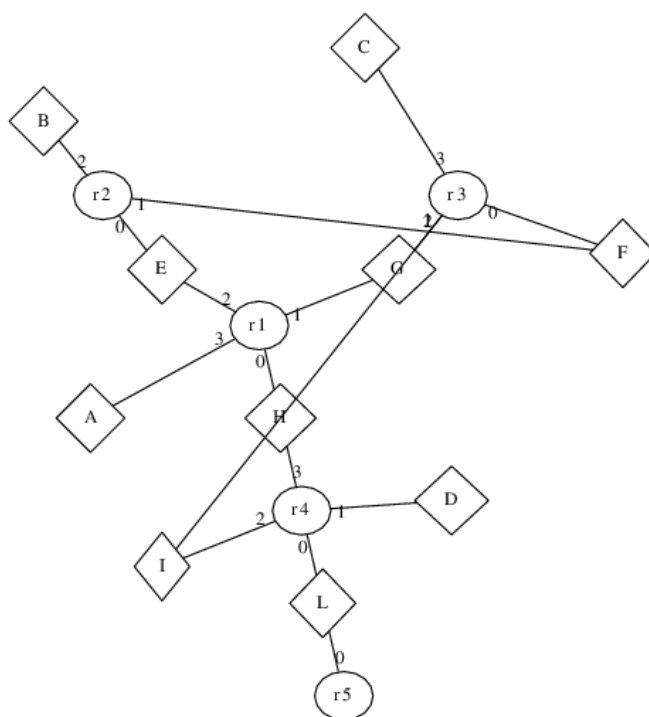
Lo primero a realizar fue instalar las herramientas necesarias para llevar a cabo el laboratorio.

Para ellos llevamos a cabo los siguientes pasos:

1. Descargamos la imagen iso de NetKit y los archivos correspondientes a VMWare para crear el entorno de trabajo.
2. Iniciamos la maquina virtual y levantamos un servidor ssh para la transmisión de archivos desde el sistema operativo real
3. Copiamos el archivo netkit-lab\_rip.tar.gz y lo descomprimos.
4. Por último ejecutamos ltest para verificar su correctitud, obteniendo la respuesta "test succeeded".

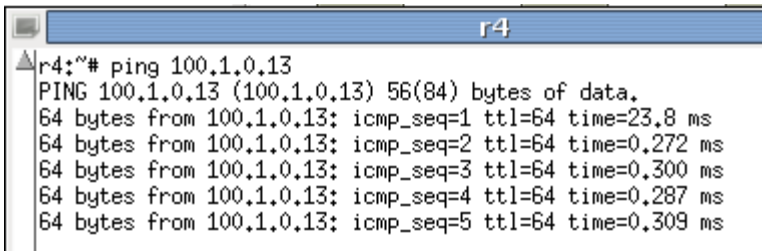
Antes de comenzar con la ejecución del laboratorio, nos pareció interesante explorar la representación grafica que nos brinda el comando linfo. Para ello hubo que instalar graphvis siguiendo los siguientes pasos:

1. Ejecutamos **sudo apt-get update** para actualizar las direcciones de descarga.
2. Ejecutamos **sudo apt-get install graphviz** para instalar la librería graphviz.
3. Luego parados en el directorio del laboratorio en cuestión, ejecutamos **linfo -m nombreGrafica.ps** para generar un fichero postscript.
4. Lo abrimos ejecutando **gs nombreGrafica.ps** y obtenemos la siguiente topología generada con GraphViz, la cual se mapea con la topología descrita anteriormente.



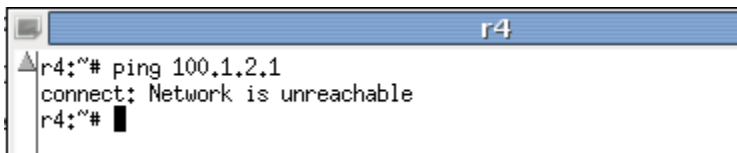
Luego de esto, iniciamos el laboratorio con la ejecución del comando **lstart** el cual ejecuta cada uno de los script **ri.startup** ( $i = 1 \dots 5$ ) y levanta cada una de las maquinas virtuales uml.

A continuación verificamos la conectividad de r4 con r1 enviando un paquete ICMP desde r4 hacia r1 (**ping 100.1.0.13**). En la siguiente imagen se puede observar que la misma es exitosa, lo cual es coherente ya que ambos routers están directamente conectados.



```
r4:~# ping 100.1.0.13
PING 100.1.0.13 (100.1.0.13) 56(84) bytes of data:
 64 bytes from 100.1.0.13: icmp_seq=1 ttl=64 time=23.8 ms
 64 bytes from 100.1.0.13: icmp_seq=2 ttl=64 time=0.272 ms
 64 bytes from 100.1.0.13: icmp_seq=3 ttl=64 time=0.300 ms
 64 bytes from 100.1.0.13: icmp_seq=4 ttl=64 time=0.287 ms
 64 bytes from 100.1.0.13: icmp_seq=5 ttl=64 time=0.309 ms
```

Luego chequeamos la conectividad de r4 con r2 (**ping 100.1.2.1**), a lo cual se obtiene la respuesta “Network is unreachable” que es coherente ya que ambos routers no están directamente conectados y aun no hemos levantado el demonio zebra para que los routers se conozcan.



```
r4:~# ping 100.1.2.1
connect: Network is unreachable
r4:~#
```

### b) Captura de paquetes en las interfaces de r1

Con “**tcpdump -i any -w /home/ripd.startup.r1.cap &**” en la terminal correspondiente a r1 iniciamos la captura de paquetes antes de levantar el demonio zebra, liberando la terminal con &.

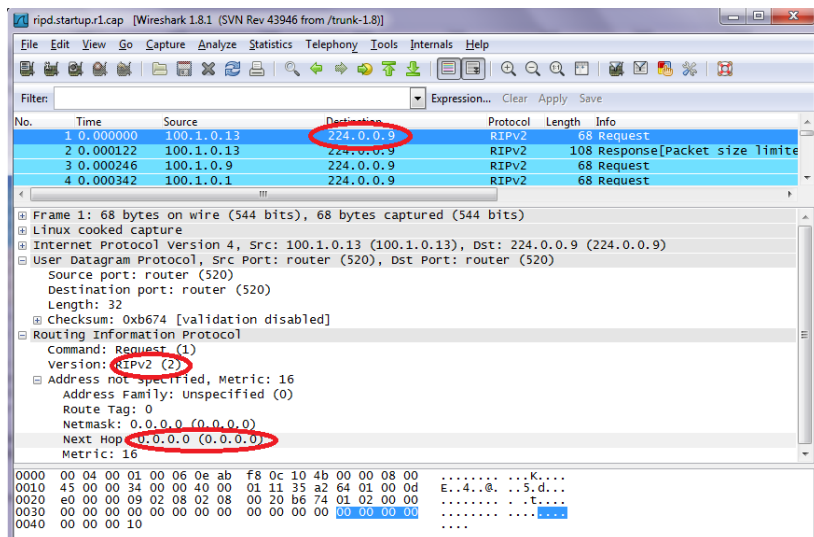
Análisis de los parámetros de tcpdump:

**-i** indica que interfaces queremos capturar, al decirle “**any**” las toma todas.

**-w nombreArchivo** nos permite indicarle el archivo de salida de la captura.

Luego levantamos el demonio de zebra en cada terminal con “**/etc/init.d/zebra start**” y esperamos 30 segundos para cerrar el archivo de captura.

## c) Análisis del archivo ripd.startup.r1.cap



El protocolo RIP maneja dos tipos de mensajes:

- Request: Utilizados para solicitar una copia de toda o parte de la tabla de encaminamiento.
- Response: Utilizados para actualizar las tablas de encaminamiento. Hay 3 tipos:
  - Mensajes que se envían cada 30 segundos para mostrar que el enlace y la ruta siguen activos.
  - Mensajes enviados como respuesta a mensajes Request.
  - Mensajes enviados cuando cambia algún costo.

## 1. Mensajes Request

El contenido de los mensajes Request de RipV2 es el siguiente:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Command								Version								Unused																0
Address Family																Route Tag																1
IPv4 Address																																2
Subnet Mask																																3
Next Hop																																4
Metric																																5

Dónde:

- Comando (1 byte): vale 0x01, indicando que es un mensaje del tipo Request.
- Versión (1 byte): vale 0x02, indicando que es RipV2.
- Datos de la tabla de entradas de RipV2 (si corresponde):
  - Address family (2 bytes):  
Especifica la familia de direcciones utilizada. Si el mensaje es un Request de una tabla de enrutamiento completa, se carga con 0x0, sino valdrá 0x2 para indicar que usaremos una dirección IP.
  - Route tag (2 bytes):  
Este es un atributo asignado a una ruta que debe ser preservada. Se utiliza para distinguir entre rutas internas (rutas de las redes en el dominio de enrutamiento RIP) y rutas externas (rutas aprendidas de otros protocolos de enrutamiento).
  - IPv4 address (4 bytes):  
Especifica la dirección IP destino del mensaje.
  - Subnet mask (4 bytes):  
Contiene la máscara de subred que se debe aplicar a la dirección IPv4 para resolver la red de esa dirección. Si el campo se pone a 0, no se especifica la máscara de subred.
  - Next hop (4 bytes):  
Indica la dirección IP a donde los paquetes para esta entrada deberían de ser transferidos.
  - Metric (4 bytes):  
Contiene un valor del 1 al 15 que indica la actual métrica para el destino. El valor 16 indica que el destino no es alcanzable.

En la imagen visualizada con el wireshark de la captura de paquetes (ripd.startup.r1.cap) se pueden visualizar todos los campos mencionados.

## 2. Dirección de destino de los mensajes Request

Como se observa en la imagen *ripd.startup.r1.cap* la dirección de destino de los mensajes de Request es **224.0.0.9**. Esta es una dirección multicast particular utilizada por el RIPv2 para enviar información de routing a todos los routers RIP2 en un segmento de red. Un router RIP envía mensajes a sus adyacentes utilizando esa dirección multicast. Utilizando esta dirección los routers RIP pueden coexistir con los routers que no son RIP puesto que estos estarían asociados a otra dirección multicast. La ventaja de este multicasting es que disminuye la escucha innecesaria por parte de los dispositivos que no escuchan mensajes RIP.



### 3. Next Hop en un Response

Este campo identifica la dirección IP a la cual el mensaje debe enviarse para obtener la mejor ruta, esta dirección deberá ser de un router adyacente al router origen.

En el caso de un mensaje del tipo Response, este valor es 0.0.0.0. Esto indica que el ruteo lo efectúa el router en cuestión y no a través de la subred lógica indicada. Es decir, el próximo salto en la ruta será el router que envió el paquete Response.

### 4. Visualización de la información de mensajería de RIP con el debug del demonio ripd en R1.

Existen 3 modalidades de debug posibles de implementar:

- debug rip events: Muestra los eventos de RIP
- debug rip packet: Muestra información detallada sobre los paquetes de RIP.
- debug rip zebra: Muestra la comunicación entre ripd y zebra.

Por lo que para visualizar la información de mensajería de RIP debemos utilizar el modo de paquetes (*debug rip packet*), sin embargo también habilitaremos el debug de eventos para obtener mayor información.

Para habilitarlos tenemos dos formas de hacerlo:

La primera es modificando el archivo ripd.conf agregando las líneas “debug rip events” y “debug rip packet” y reiniciando el demonio para que los cambios surjan efecto.

La segunda, que fue por la que optamos, es conectándose por telnet contra el demonio ripd y habilitarlos. Esto es posible porque ripd es VTY (Virtual Teletype interface).

Utilizando los comandos de debug de ripd, levantamos los modos debug necesarios (events y packet) y cambiando la configuración indicamos en que archivo realizar la captura.

En nuestro primer intento nos encontramos con que debemos crear y dar permisos al archivo donde vamos a realizar la captura, por lo que primero realizamos esta creación.

Por lo tanto parados en R1 seguimos los siguientes pasos:

- 1- Creamos el archivo de log
- 2- Otorgamos los permisos correspondientes.
- 3- Nos conectamos al demonio rip y habilitamos los debug necesarios

A continuación se muestra en detalle los pasos seguidos:

```
r4:~# touch /hosthome/ripd.log.r1.pcap      ! Creamos el archivo de log
r4:~# chmod 777 /hosthome/ripd.log.r1.pcap ! Otorgamos permisos
r4:~# telnet localhost ripd                ! Nos conectamos al demonio rip
Trying 127.0.0.1...
Connected to r4.
Escape character is '^]'.
Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.
User Access Verification
Password: zebra ! ingresamos password
ripd> enable ! habilitamos usuario con privilegios
Password: zebra ! ingresamos password
ripd# debug rip packet ! habilitamos el debug de paquetes
ripd# debug rip events ! habilitamos el debug de eventos
ripd# configure terminal ! ingresamos a modo configuración
ripd(config)# log file /hosthome/ripd.log.r1.pcap ! indicamos el archive de salida
ripd(config)# quit ! cerramos modo configuración
ripd# disable ! deshabilitamos usuario con privilegios
ripd> exit ! cerramos conexión telnet
Connection closed by foreign host.
```

Luego observando el archivo de salida vemos por ejemplo cuando vence el temporizador de actualización (**update timer fire!**) y se envían las mismas (**SEND UPDATE**).

El protocolo RIP maneja básicamente tres temporizadores:

- Un temporizador de actualización (update timer) el cual utiliza para saber cuándo enviar una actualización.
- Un temporizador de validez (timer) el cual se utiliza para saber cuándo una ruta ya no es válida.
- Un temporizador del recolector de basura (garbage-collection) el cual se utiliza para saber cuándo eliminar definitivamente una ruta.

Los valores por defecto de los temporizadores son los siguientes:

- El timeout del update timer es cada 30 segundos. Cada vez que se llega al timeout, se produce una actualización, el proceso RIP es despertado para enviar un mensaje de respuesta no solicitado conteniendo la tabla completa de forwarding a todos los routers vecinos.
- El timeout del temporizador de validez tiene un valor por defecto de 180 segundos. Cuando llega una actualización el timer vuelve a cero, por lo que el vencimiento se da cuando pasan 180 segundos sin noticias del nodo. Si vence este timeout, la ruta se considera no válida pero igualmente se mantiene (con métrica 16) un cierto periodo de tiempo en la tabla, lo cual permite avisarle a los vecinos que la ruta ha sido eliminada.
- El timeout del temporizador del recolector de basura vale por defecto 120 segundos. Cuando se llega a este timeout, la ruta es definitivamente eliminada de la tabla de forwarding.

Mirando la salida del debug también se observa que la dirección de destino de los mensajes de Request es la dirección de multicast 224.0.0.9, como se explico en el punto 2.

También se confirma que el puerto utilizado por RIP para enviar y recibir datagramas UDP es el 520.

Además, con esta salida también se observan los tamaños de los paquetes intercambiados entre los routers y las métricas utilizadas.

RIP utiliza una métrica fija para comparar las distintas rutas que es la cantidad de routers por los que debe pasar un paquete para llegar a su destino. Por ello en RIP la métrica a las redes directamente conectadas es 1.

## d) Continuación del laboratorio

### 1- Verificando conectividad

Luego de haber levantado zebra en cada uno de los routers y esperar 30 segundos, volvemos a chequear la conectividad de r4 con r2 (ping 100.1.2.1), a lo cual ahora si se obtiene una respuesta exitosa. Verificamos que su tabla de routing se actualizo con el comando **route**.

```
r4:~# ping 100.1.2.1
PING 100.1.2.1 (100.1.2.1) 56(84) bytes of data:
64 bytes from 100.1.2.1: icmp_seq=1 ttl=63 time=22.0 ms
64 bytes from 100.1.2.1: icmp_seq=2 ttl=63 time=0.881 ms
64 bytes from 100.1.2.1: icmp_seq=3 ttl=63 time=0.684 ms
64 bytes from 100.1.2.1: icmp_seq=4 ttl=63 time=0.695 ms
^C
--- 100.1.2.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3026ms
rtt min/avg/max/mdev = 0.684/6.089/22.098/9.243 ms
r4:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
100.1.0.16 * 255.255.255.252 U 0 0 0 eth2
100.1.0.0 100.1.0.13 255.255.255.252 UG 2 0 0 eth3
100.1.0.4 100.1.0.17 255.255.255.252 UG 2 0 0 eth2
100.2.0.0 * 255.255.255.252 U 0 0 0 eth0
100.1.0.8 100.1.0.13 255.255.255.252 UG 2 0 0 eth3
100.1.0.12 * 255.255.255.252 U 0 0 0 eth3
100.1.4.0 * 255.255.255.0 U 0 0 0 eth1
100.1.2.0 100.1.0.13 255.255.255.0 UG 3 0 0 eth3
100.1.3.0 100.1.0.17 255.255.255.0 UG 2 0 0 eth2
100.1.1.0 100.1.0.13 255.255.255.0 UG 2 0 0 eth3
```

### 2- Sniffing con Tcpdump

Podemos hacer una captura de paquetes con el comando tcpdump.

```
r4:~# tcpdump -i eth3 -s 1500
length 172) 100.1.0.18,520 > 224.0.0.9,520:
RIPv2, Response, length: 144, routes: 7
AFI: IPv4: 100.1.0.0/30, tag 0x0000, metric: 2, next-hop: self
AFI: IPv4: 100.1.0.8/30, tag 0x0000, metric: 2, next-hop: self
AFI: IPv4: 100.1.0.12/30, tag 0x0000, metric: 1, next-hop: self
AFI: IPv4: 100.1.1.0/24, tag 0x0000, metric: 2, next-hop: self
AFI: IPv4: 100.1.2.0/24, tag 0x0000, metric: 3, next-hop: self
AFI: IPv4: 100.1.4.0/24, tag 0x0000, metric: 1, next-hop: self
AFI: IPv4: 100.2.0.0/30, tag 0x0000, metric: 1, next-hop: self
21:28:16.173954 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17),
length 172) 100.1.0.17,520 > 224.0.0.9,520:
RIPv2, Response, length: 144, routes: 7
AFI: IPv4: 100.1.0.0/30, tag 0x0000, metric: 2, next-hop: self
AFI: IPv4: 100.1.0.4/30, tag 0x0000, metric: 1, next-hop: self
AFI: IPv4: 100.1.0.8/30, tag 0x0000, metric: 1, next-hop: self
AFI: IPv4: 100.1.0.12/30, tag 0x0000, metric: 2, next-hop: self
AFI: IPv4: 100.1.1.0/24, tag 0x0000, metric: 2, next-hop: self
AFI: IPv4: 100.1.2.0/24, tag 0x0000, metric: 2, next-hop: self
AFI: IPv4: 100.1.3.0/24, tag 0x0000, metric: 1, next-hop: self
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
r4:~#
```

### 3 – Traceroute a R2

Al ejecutar un traceroute vemos que la distancia de r4 a r2 es 2, el paquete ICMP primero pasa por r3 (100.1.0.17) y luego llega a r2 (100.1.2.1)

```
r4:~# traceroute 100.1.2.1
traceroute to 100.1.2.1 (100.1.2.1), 64 hops max, 40 byte packets
1 100.1.0.17 (100.1.0.17) 9 ms 0 ms 0 ms
2 100.1.2.1 (100.1.2.1) 9 ms 0 ms 0 ms
r4:~#
```

### 4 - Conectando con Zebra por telnet

Nos conectamos mediante telnet al demonio zebra.

```
r4:~# telnet localhost zebra
Trying 127.0.0.1...
Connected to r4.
Escape character is '^'.

Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
zebra>
zebra>
zebra> echo Echo a message back to the vty
zebra> enable Turn on privileged mode command
zebra> exit Exit current mode and down to previous mode
zebra> help Description of the interactive help system
zebra> list Print command list
zebra> quit Exit current mode and down to previous mode
zebra> show Show running system information
zebra> terminal Set terminal line parameters
zebra> who Display who is on vty
```

## 5 - Inspección de interface

Inspeccionamos la interfaz eth0 con el comando **show interface eth0**.

```
r4
show mpls forwarding
show mpls hardware
show mpls version
show table
show thread cpu [FILTER]
show version
show work-queues
terminal length <0-512>
terminal no length
who
zebra> show interface eth0
Interface eth0 is up, line protocol detection is disabled
index 3 metric 1 mtu 1500
flags: <UP,BROADCAST,RUNNING,MULTICAST>
HWaddr: 4a:e4:d9:3b:f2:04
inet 100.2.0.1/30 broadcast 100.2.0.3
inet6 fe80::48e4:d9ff:fe3b:f204/64
  6 input packets, 0 multicast), 384 bytes, 0 dropped
  0 input errors, 0 length, 0 overrun, 0 CRC, 0 frame
  0 fifo, 0 missed
  6 output packets, 468 bytes, 0 dropped
  0 output errors, 0 aborted, 0 carrier, 0 fifo, 0 heartbeat
  0 window, 0 collisions
zebra>
```

## 6 - Inspección de la tabla de enrutamiento de zebra

Inspeccionamos la tabla de routing de zebra con el comando **show ip route**

```
r4
zebra> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
I - ISIS, B - BGP, > - selected route, * - FIB route

R>* 100.1.0.0/30 [120/2] via 100.1.0.13, eth3, 08:52:01
R>* 100.1.0.4/30 [120/2] via 100.1.0.17, eth2, 08:52:01
R>* 100.1.0.8/30 [120/2] via 100.1.0.17, eth2, 08:52:01
C>* 100.1.0.12/30 is directly connected, eth3
C>* 100.1.0.16/30 is directly connected, eth2
R>* 100.1.1.0/24 [120/2] via 100.1.0.13, eth3, 08:52:01
R>* 100.1.2.0/24 [120/3] via 100.1.0.17, eth2, 08:52:01
R>* 100.1.3.0/24 [120/2] via 100.1.0.17, eth2, 08:52:01
C>* 100.1.4.0/24 is directly connected, eth1
C>* 100.2.0.0/30 is directly connected, eth0
C>* 127.0.0.0/8 is directly connected, lo
zebra>
```

## 7 - Configurando Zebra

```
r4
R>* 100.1.0.0/30 [120/2] via 100.1.0.13, eth3, 05:19:35
R>* 100.1.0.4/30 [120/2] via 100.1.0.17, eth2, 05:19:35
R>* 100.1.0.8/30 [120/2] via 100.1.0.13, eth3, 05:19:35
C>* 100.1.0.12/30 is directly connected, eth3
C>* 100.1.0.16/30 is directly connected, eth2
R>* 100.1.1.0/24 [120/2] via 100.1.0.13, eth3, 05:19:35
R>* 100.1.2.0/24 [120/3] via 100.1.0.13, eth3, 05:19:35
R>* 100.1.3.0/24 [120/2] via 100.1.0.17, eth2, 05:19:35
C>* 100.1.4.0/24 is directly connected, eth1
C>* 100.2.0.0/30 is directly connected, eth0
C>* 127.0.0.0/8 is directly connected, lo
zebra> enable
Password:
zebra# configure terminal
zebra(config)# hostname zebra-r4
zebra-r4(config)# password foo
zebra-r4(config)# quit
zebra-r4# write file
Configuration saved to /etc/quagga/zebra.conf
zebra-r4# disable
zebra-r4> exit
Connection closed by foreign host.
r4:~#
```

## 8 - Inspección de la tabla de enrutamiento de RIP

```
r4
ripd> show ip rip
Codes: R - RIP, C - connected, S - Static, O - OSPF, B - BGP
Sub-codes:
  (n) - normal, (s) - static, (d) - default, (r) - redistribute,
  (i) - interface

Network          Next Hop          Metric From          Tag Time
R(n) 100.1.0.0/30 100.1.0.13        2 100.1.0.13          0 02:26
R(n) 100.1.0.4/30 100.1.0.17        2 100.1.0.17          0 02:41
R(n) 100.1.0.8/30 100.1.0.17        2 100.1.0.17          0 02:41
C(i) 100.1.0.12/30 0.0.0.0           1 self               0
C(i) 100.1.0.16/30 0.0.0.0           1 self               0
R(n) 100.1.1.0/24 100.1.0.13        2 100.1.0.13          0 02:26
R(n) 100.1.2.0/24 100.1.0.17        3 100.1.0.17          0 02:41
R(n) 100.1.3.0/24 100.1.0.17        2 100.1.0.17          0 02:41
C(i) 100.1.4.0/24 0.0.0.0           1 self               0
C(r) 100.2.0.0/30 0.0.0.0           1 self (connected:1) 0
ripd>
```

## 9 - Agregando ruta estática a R5

```
r5:~# ping 100.1.2.1
connect: Network is unreachable
r5:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
100.2.0.0 * 255.255.255.252 U 0 0 0 eth0
r5:~# route add -net 100.1.0.0/16 gw 100.2.0.1
r5:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
100.2.0.0 * 255.255.255.252 U 0 0 0 eth0
100.1.0.0 100.2.0.1 255.255.0.0 UG 0 0 0 eth0
r5:~# ping -c 2 100.1.2.1
PING 100.1.2.1 (100.1.2.1) 56(84) bytes of data:
64 bytes from 100.1.2.1: icmp_seq=1 ttl=62 time=0.667 ms
64 bytes from 100.1.2.1: icmp_seq=2 ttl=62 time=0.538 ms

--- 100.1.2.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1017ms
rtt min/avg/max/mdev = 0.538/0.602/0.667/0.069 ms
r5:~#
```

## 10- Configurar ruta por defecto en R4

```
r4:~# route add default gw 100.2.0.2
r4:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
100.1.0.16 * 255.255.255.252 U 0 0 0 eth2
100.1.0.0 100.1.0.13 255.255.255.252 UG 2 0 0 eth3
100.1.0.4 100.1.0.17 255.255.255.252 UG 2 0 0 eth2
100.2.0.0 * 255.255.255.252 U 0 0 0 eth0
100.1.0.8 100.1.0.17 255.255.255.252 UG 2 0 0 eth2
100.1.0.12 * 255.255.255.252 U 0 0 0 eth3
100.1.4.0 * 255.255.255.0 U 0 0 0 eth1
100.1.2.0 100.1.0.17 255.255.255.0 UG 3 0 0 eth2
100.1.3.0 100.1.0.17 255.255.255.0 UG 2 0 0 eth2
100.1.1.0 100.1.0.13 255.255.255.0 UG 2 0 0 eth3
default 100.2.0.2 0.0.0.0 UG 0 0 0 eth0
r4:~#
```

## 11- Propagamos la ruta por defecto

```
r4~# telnet localhost ripd
Trying 127.0.0.1...
Connected to r4.
Escape character is '^'.

Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
ripd> enable
Password:
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# route 0.0.0.0/0
ripd(config-router)# quit
ripd(config)# quit
ripd# disable
ripd> exit
Connection closed by foreign host.
r4~#
```

## 12- Verificamos la propagación

```
r1~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
100.1.0.16 100.1.0.10 255.255.255.252 UG 2 0 0 eth1
100.1.0.0 * 255.255.255.252 U 0 0 0 eth2
100.2.0.0 100.1.0.14 255.255.255.252 UG 2 0 0 eth0
100.1.0.4 100.1.0.2 255.255.255.252 UG 2 0 0 eth2
100.1.0.8 * 255.255.255.252 U 0 0 0 eth1
100.1.0.12 * 255.255.255.252 U 0 0 0 eth0
100.1.4.0 100.1.0.14 255.255.255.0 UG 2 0 0 eth0
100.1.2.0 100.1.0.2 255.255.255.0 UG 2 0 0 eth2
100.1.3.0 100.1.0.10 255.255.255.0 UG 2 0 0 eth1
100.1.1.0 * 255.255.255.0 U 0 0 0 eth3
default 100.1.0.14 0.0.0.0 UG 2 0 0 eth0
r1~# ping 193.204.161.1
PING 193.204.161.1 (193.204.161.1) 56(84) bytes of data.
From 100.2.0.2 icmp_seq=1 Destination Net Unreachable
From 100.2.0.2 icmp_seq=2 Destination Net Unreachable
From 100.2.0.2 icmp_seq=3 Destination Net Unreachable
^C
--- 193.204.161.1 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2010ms
```

## 13- Recepción de echo request en R5

Se observa que al enviar un paquete desde R1 hacia fuera de la red (193.204.161.1), el mismo le llega a R5. Sin embargo en ping en R1 indica que esa red es inalcanzable (193.204.161.1) ya que R5 no está configurado para responder a esa dirección IP.

```
r1~# ping -c 2 193.204.161.1
PING 193.204.161.1 (193.204.161.1) 56(84) bytes of data.
From 100.2.0.2 icmp_seq=1 Destination Net Unreachable
From 100.2.0.2 icmp_seq=2 Destination Net Unreachable

--- 193.204.161.1 ping statistics ---
2 packets transmitted, 0 received, +2 errors, 100% packet loss, time 1016ms
r1~#

r5~# tcpdump -i eth0 -n -s 1518
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 1518 bytes
06:29:35.056733 arp who-has 100.2.0.2 tell 100.2.0.1
06:29:35.057835 arp reply 100.2.0.2 is-at 26:55:90:f3:dd:97
06:29:35.056807 IP 100.1.0.13 > 193.204.161.1: ICMP echo request, id 52226, seq 1, length 64
06:29:35.056936 IP 100.2.0.2 > 100.1.0.13: ICMP net 193.204.161.1 unreachable, length 92
06:29:36.057044 IP 100.1.0.13 > 193.204.161.1: ICMP echo request, id 52226, seq 2, length 64
06:29:36.057092 IP 100.2.0.2 > 100.1.0.13: ICMP net 193.204.161.1 unreachable, length 92
06:29:40.048997 arp who-has 100.2.0.1 tell 100.2.0.2
06:29:40.049203 arp reply 100.2.0.1 is-at 4a:e4:d9:3b:f2:04
^C
8 packets captured
8 packets received by filter
0 packets dropped by kernel
```

## 14- Eliminación de eth1 en R1

Parados en R1, al hacer un traceroute a R3 llegamos en un solo salto ya que están directamente conectados.

Luego, si apagamos ese enlace directo y volvemos a hacer enseguida un traceroute a R3, R1 no encuentra esa ruta e intenta enviar el paquete a través de la ruta por defecto, que en nuestro caso se dirige a R5.

Después de un cierto tiempo, la información de routing se distribuye entre los routers y R1 vuelve a conocer donde se encuentra R3, solo que ahora tiene que realizar un salto mas y pasar por R4 (100.1.0.14).

```
r1:~# traceroute 100.1.0.10
traceroute to 100.1.0.10 (100.1.0.10), 64 hops max, 40 byte packets
 1 100.1.0.10 (100.1.0.10) 1 ms 0 ms 0 ms
r1:~# ifconfig eth1 down
r1:~# traceroute 100.1.0.10
traceroute to 100.1.0.10 (100.1.0.10), 64 hops max, 40 byte packets
 1 100.1.0.14 (100.1.0.14) 14 ms 0 ms 0 ms
 2 * * *
 3 * * *
 4 * * *
 5 ^C
r1:~# traceroute 100.1.0.10
traceroute to 100.1.0.10 (100.1.0.10), 64 hops max, 40 byte packets
 1 100.1.0.2 (100.1.0.2) 10 ms 0 ms 0 ms
 2 100.1.0.10 (100.1.0.10) 11 ms 0 ms 0 ms
r1:~#
```

```
r1:~# traceroute 100.1.0.10
traceroute to 100.1.0.10 (100.1.0.10), 64 hops max, 40 byte packets
 1 100.1.0.14 (100.1.0.14) 1 ms 0 ms 0 ms
 2 100.1.0.10 (100.1.0.10) 14 ms 1 ms 0 ms
r1:~# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
100.1.0.16 100.1.0.14 255.255.255.252 UG 2 0 0 eth0
100.1.0.0 * 255.255.255.252 U 0 0 0 eth2
100.2.0.0 100.1.0.14 255.255.255.252 UG 2 0 0 eth0
100.1.0.4 100.1.0.2 255.255.255.252 UG 2 0 0 eth2
100.1.0.8 100.1.0.14 255.255.255.252 UG 3 0 0 eth0
100.1.0.12 * 255.255.255.252 U 0 0 0 eth0
100.1.4.0 100.1.0.14 255.255.255.0 UG 2 0 0 eth0
100.1.2.0 100.1.0.2 255.255.255.0 UG 2 0 0 eth2
100.1.3.0 100.1.0.2 255.255.255.0 UG 3 0 0 eth2
100.1.1.0 * 255.255.255.0 U 0 0 0 eth3
default 100.1.0.14 0.0.0.0 UG 2 0 0 eth0
r1:~#
```

## e) Conexión a zebra en R3 mediante telnet.

Next Hop para las redes 100.1.0.12/30 y 100.1.0.0/30.

Identificación de distancia administrativa y la métrica [120/2].

El next hop lo identificamos mirando en “via 100.1.0.18” que indica la dirección del router para el próximo salto.

Para 100.1.0.12/30 el próximo salto es 100.1.0.18

Para 100.1.0.0/30 el próximo salto es 100.1.0.5

[120/2] significa la distancia administrativa y la distancia métrica.

En RIP la distancia administrativa por defecto es 120 y el 2 significa que hay 2 hops entre origen y destino.

```
r3
Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
zebra> sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       I - ISIS, B - BGP, > - selected route, * - FIB route
R>* 0.0.0.0/0 [120/2] via 100.1.0.10, eth0, 00:00:57
R>* 100.1.0.0/30 [120/2] via 100.1.0.5, eth0, 05:59:30
C>* 100.1.0.4/30 is directly connected, eth0
C>* 100.1.0.8/30 is directly connected, eth1
R>* 100.1.0.12/30 [120/2] via 100.1.0.18, eth2, 00:08:51
C>* 100.1.0.16/30 is directly connected, eth2
R>* 100.1.1.0/24 [120/3] via 100.1.0.18, eth2, 00:08:51
R>* 100.1.2.0/24 [120/2] via 100.1.0.5, eth0, 05:59:30
C>* 100.1.3.0/24 is directly connected, eth3
R>* 100.1.4.0/24 [120/2] via 100.1.0.18, eth2, 05:59:12
R>* 100.2.0.0/30 [120/2] via 100.1.0.18, eth2, 05:59:12
C>* 127.0.0.0/8 is directly connected, lo
zebra>
```

La distancia administrativa califica la confiabilidad de la información de enrutamiento, se utiliza como criterio de selección cuando el dispositivo tiene múltiples rutas hacia el mismo destino. La mejor ruta es la que tenga menor distancia administrativa. Esto sucede ya que un router puede ejecutar varios protocolos de enrutamiento a la vez, obteniendo información de una red por varias fuentes.

La distancia métrica es la cantidad de hops para llegar a destino.

### f) ¿Por qué es necesario agregar una ruta estática a 100.1.0.0/16 en r5?

Es necesario para alcanzar desde r5 a la subred 100.1.0.0/16. O sea, siendo r5 un router externo a nuestra red es necesario indicarle el Gateway para acceder a 100.1.0.0/16 a través de r4.

En el laboratorio se agregó dicha ruta estática ejecutando el comando **route add -net 100.1.0.0/16 gw 100.2.0.1**

Explicación del comando:

- **route add** modifica el kernel agregando una entrada a la tabla de encaminamiento del router.
- **net** indica que el objetivo es una red (100.1.0.0/16)
- **gw** indica el Gateway por el cual se rutearan los paquetes (100.2.0.1)

### g) ¿Por qué se instala una ruta por defecto en r4? ¿Por qué se redistribuye en RIP?

Se instala una ruta por defecto en r4 para que el tráfico que reciba el router que no sea para la red la reenvíe por dicha ruta. La instalación fue realizada en el punto 10 del laboratorio con el comando **route add default gw 100.2.0.2**.

Con esto todos los paquetes que reciba R4 con destino fuera de la red 100.1.0.0/16, serán enviados a R5 a través de su interfaz eth0.

Se redistribuye para que los demás routers conozcan la ruta por defecto, cargándola en sus tablas de ruteo, y así puedan rutear paquetes con destino fuera de la red 100.1.0.0/16.

El protocolo RIP no distribuye de forma automática estas rutas, por ello debimos realizar la siguiente configuración para que esto suceda:

```
r4:~# telnet localhost ripd
Trying 127.0.0.1...
Connected to r4.
Escape character is '^]'.
Hello, this is zebra (version 0.94).
Copyright 1996-2002 Kunihiro Ishiguro.
User Access Verification
Password: zebra
ripd> enable
Password: zebra
ripd# configure terminal
ripd(config)# router rip
ripd(config-router)# route 0.0.0.0/0
ripd(config-router)# quit
ripd(config)# quit
ripd# disable
ripd> exit
```



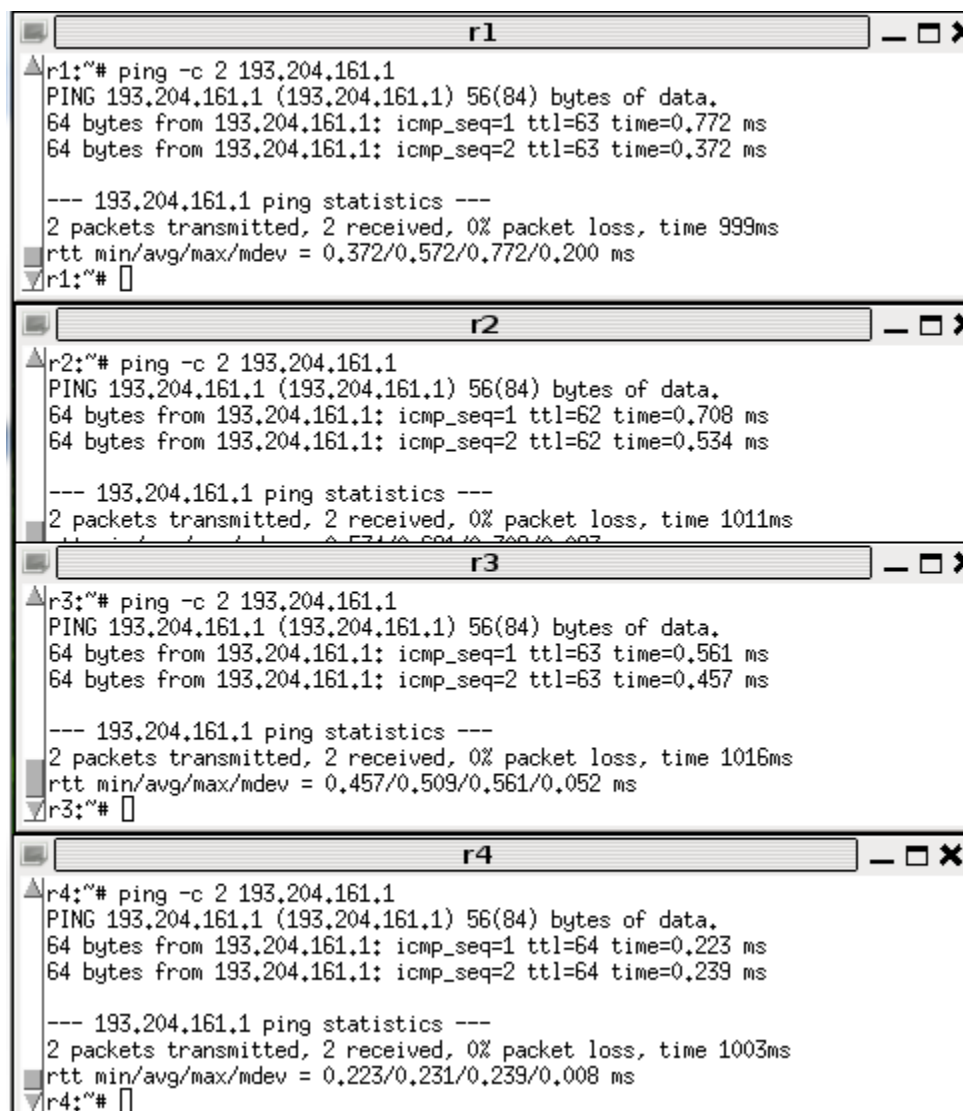
### h) Comando *ping 193.204.161.1* exitoso en routers r1...r4

Como vimos anteriormente los paquetes ICMP que los routers r1...r4 envían a 193.204.161.1 le llegan al router r5 ya que todos tienen configurada su ruta por defecto a r5. Para que el ping sea exitoso alcanza con hacer que r5 responda a esa dirección pública. Para esto debemos crear un alias a una interfaz de r5 con esa dirección IP.

Nos pareció lo más correcto hacerlo en la interfaz de loopback ya que esta es una interfaz lógica ya existente y no requiere hardware adicional.

Para implementar esta modificación alcanza con pararse en r5 y ejecutar el comando *ifconfig lo:0 193.204.161.1*

En la siguiente imagen se puede ver como ahora la ejecución de ese ping es exitosa en cada uno de los router de nuestra red.



```
r1
r1:~# ping -c 2 193.204.161.1
PING 193.204.161.1 (193.204.161.1) 56(84) bytes of data.
64 bytes from 193.204.161.1: icmp_seq=1 ttl=63 time=0.772 ms
64 bytes from 193.204.161.1: icmp_seq=2 ttl=63 time=0.372 ms

--- 193.204.161.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.372/0.572/0.772/0.200 ms
r1:~#

r2
r2:~# ping -c 2 193.204.161.1
PING 193.204.161.1 (193.204.161.1) 56(84) bytes of data.
64 bytes from 193.204.161.1: icmp_seq=1 ttl=62 time=0.708 ms
64 bytes from 193.204.161.1: icmp_seq=2 ttl=62 time=0.534 ms

--- 193.204.161.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1011ms
rtt min/avg/max/mdev = 0.534/0.604/0.708/0.097 ms
r2:~#

r3
r3:~# ping -c 2 193.204.161.1
PING 193.204.161.1 (193.204.161.1) 56(84) bytes of data.
64 bytes from 193.204.161.1: icmp_seq=1 ttl=63 time=0.561 ms
64 bytes from 193.204.161.1: icmp_seq=2 ttl=63 time=0.457 ms

--- 193.204.161.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1016ms
rtt min/avg/max/mdev = 0.457/0.509/0.561/0.052 ms
r3:~#

r4
r4:~# ping -c 2 193.204.161.1
PING 193.204.161.1 (193.204.161.1) 56(84) bytes of data.
64 bytes from 193.204.161.1: icmp_seq=1 ttl=64 time=0.223 ms
64 bytes from 193.204.161.1: icmp_seq=2 ttl=64 time=0.239 ms

--- 193.204.161.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.223/0.231/0.239/0.008 ms
r4:~#
```

## Laboratorio de Routing y Forwarding (ospf)

OSPF es un protocolo de enrutamiento jerárquico IGP que utiliza un algoritmo de estado de enlace, basado en el algoritmo de Dijkstra para calcular las rutas más optimas.

En este protocolo todos los routers de un área intercambian sus datos (LSAs) para armar un arbol SPF (shortest path first) con la topología del área, y luego según su costo arman la tabla de forwarding con las mejores rutas a los destinos del área.

**Dado el ejemplo anterior (con las modificaciones especificadas en el apartado h), cambie el Protocolo de routing de RIP a OSPF con una única área de backbone.**

**a) ¿Se puede hacer desde en línea (sin re-levantar el laboratorio)? Justifique su respuesta. Almacene los archivos del laboratorio modificado en el directorio IRC-lab\_ospf.**

Se realizan cambios sobre el archivo daemons de cada router, indicando que el demonio que se ejecutara será el correspondiente al protocolo ospf y no al rip. El archivo resultante es de la forma:

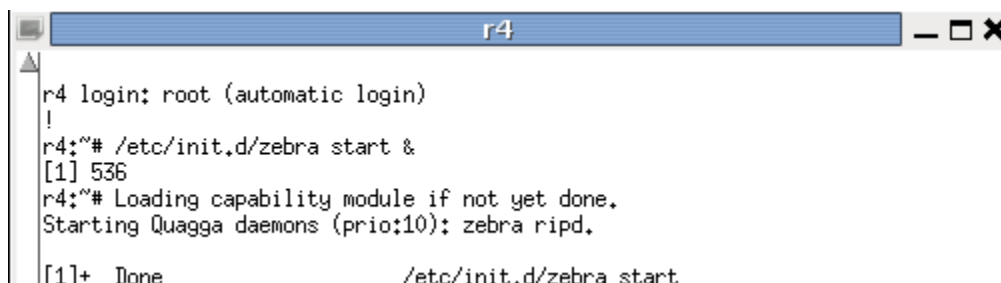
```
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

También se modifican los archivos ripd.conf, de tal forma que los mismos pasan a llamarse ospfd.conf. Se indica que el hostname es ospfd y al especificar la red, el área 0.0.0.0 es decir el área backbone, a la cual pertenecerán los routers r1..r4. Los archivos resultantes quedan de la forma:

```
hostname ospfd
password zebra
enable password zebra
!
router ospf
redistribute connected
network 100.1.0.0/16 area 0.0.0.0
!
log file /var/log/zebra/ospfd.log
```

Sobre la pregunta si dichas modificaciones se pueden realizar en línea. Consideramos que el hecho de no permanecer en línea implica realizar un *lhalt*, es decir, deteniendo las vm's inicializadas, y que la detención e inicialización de los demonios zebra no se considera una pérdida del estado en línea. Dicho esto llegamos a la conclusión de que si es posible el cambio de protocolo en línea, realizando las modificaciones detalladas anteriormente. El procedimiento consiste en:

1. Realizar */start* de forma de inicializar las VM's con la configuración del protocolo RIP.
2. Inicializar los demonios en cada una de las terminales correspondientes mediante el comando *"/etc/init.d/zebra start"* (esto se podría hacer agregando del comando de inicialización de los demonios en el archivo *.startup* para cada router).

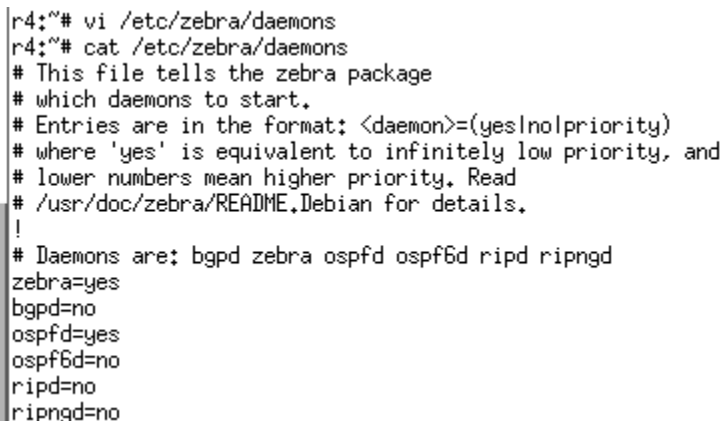


```
r4 login: root (automatic login)
!
r4:~# /etc/init.d/zebra start &
[1] 536
r4:~# Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ripd.

[1]+  Done                  /etc/init.d/zebra start
```

3. Luego en cada una de las terminales (r1...r4) se accede a los archivos de configuración *daemons* y *ospfd.conf* en el directorio */etc/zebra* de forma de realizarle, mediante el editor *vi*, las modificaciones antes mencionadas.

- Daemons:



```
r4:~# vi /etc/zebra/daemons
r4:~# cat /etc/zebra/daemons
# This file tells the zebra package
# which daemons to start.
# Entries are in the format: <daemon>=(yes|no|priority)
# where 'yes' is equivalent to infinitely low priority, and
# lower numbers mean higher priority. Read
# /usr/doc/zebra/README.Debian for details.
!
# Daemons are: bgpd zebra ospfd ospf6d ripd ripngd
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

- Ospfd.conf



```
r3:~# vi /etc/zebra/ospfd.conf
r3:~# cat /etc/zebra/ospfd.conf
!
! Zebra configuration saved from vty
!   2008/12/04 15:22:29
!
hostname ospfd
password zebra
enable password zebra
log file /var/log/zebra/ospfd.log
!
router ospf
 redistribute connected
 network 100.1.0.0/16 area 0.0.0.0
```

4. Luego detenemos los demonios que inicializamos en el paso 2 mediante el comando *"/etc/init.d/zebra stop"*.
5. Luego se intenta iniciar los demonios nuevamente mediante *"/etc/init.d/zebra start"*. Al hacer esto obtenemos un error de permisos sobre el archivo */etc/quagga/ospfd.conf*.

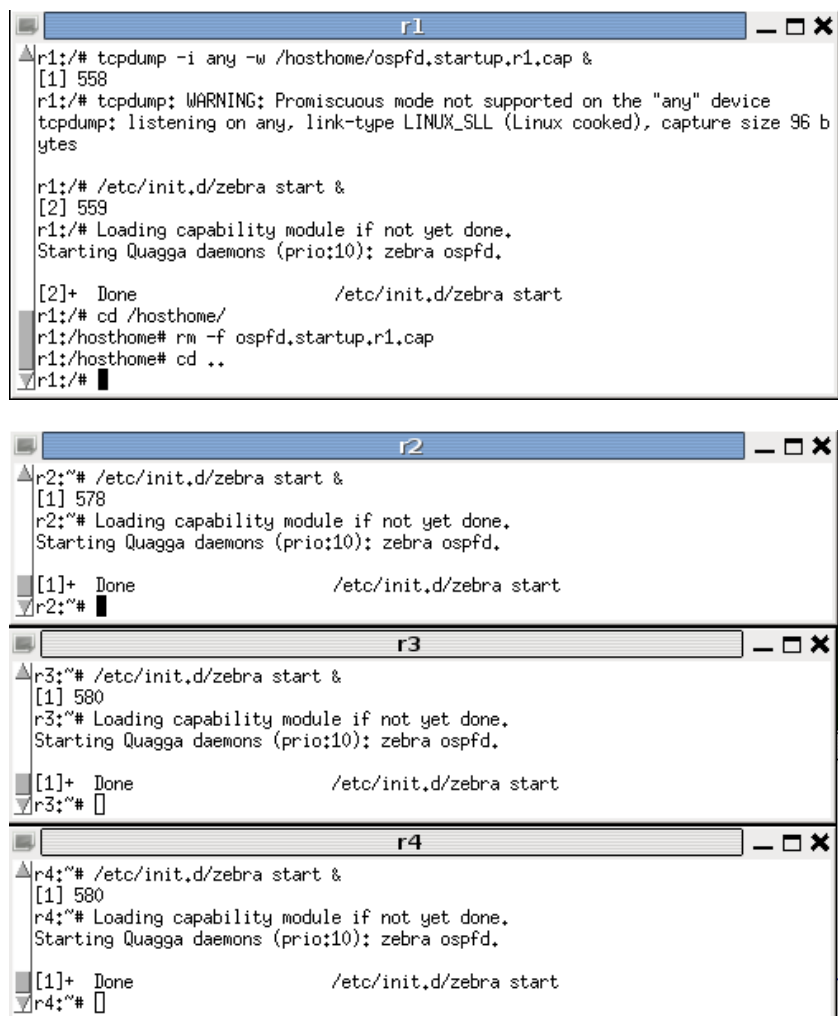
```
r4
r4:~# /etc/init.d/zebra start &
[1] 584
r4:~# Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfdvty_read_config: failed to open co
nfiguration file /etc/quagga/ospfd.conf: Permission denied
can't open configuration file [/etc/quagga/ospfd.conf]
[1]+  Exit 1                  /etc/init.d/zebra start
```

6. De forma de solucionar el problema se otorgan los permisos requeridos mediante el comando *"chmod 777"*. Luego detenemos e iniciamos los demonios que inicializamos en el paso 2 mediante los comandos *"/etc/init.d/zebra stop"* y *"/etc/init.d/zebra start"*, en cada terminal. Ahora podemos ver que el demonio que es iniciado es el correspondiente al protocolo ospf.

```
r4
r4:~# /etc/init.d/zebra stop &
[1] 600
r4:~# Stopping Quagga daemons (prio:0): (ospfd) zebra (bgpd) (ripd) (ripngd) (os
pf6d) (isisd) (ldpd).
Removing all routes made by zebra.
Nothing to flush.
[1]+  Done                  /etc/init.d/zebra stop
r4:~# chmod 777 /etc/quagga/ospfd.conf
r4:~# /etc/init.d/zebra start &
[1] 622
r4:~# Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfd.
[1]+  Done                  /etc/init.d/zebra start
r4:~#
```

### b) Continuación del laboratorio con OSPF.

### c) Captura de paquetes en las interfaces de r1 en /hosthome/ospfd.startup.r1.cap



```
r1:~# tcpdump -i any -w /hosthome/ospfd.startup.r1.cap &
[1] 558
r1:~# tcpdump: WARNING: Promiscuous mode not supported on the "any" device
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture size 96 b
ytes

r1:~# /etc/init.d/zebra start &
[2] 559
r1:~# Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfd.

[2]+ Done /etc/init.d/zebra start
r1:~# cd /hosthome/
r1:/hosthome# rm -f ospfd.startup.r1.cap
r1:/hosthome# cd ..
r1:~#
```

```
r2:~# /etc/init.d/zebra start &
[1] 578
r2:~# Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfd.

[1]+ Done /etc/init.d/zebra start
r2:~#
```

```
r3:~# /etc/init.d/zebra start &
[1] 580
r3:~# Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfd.

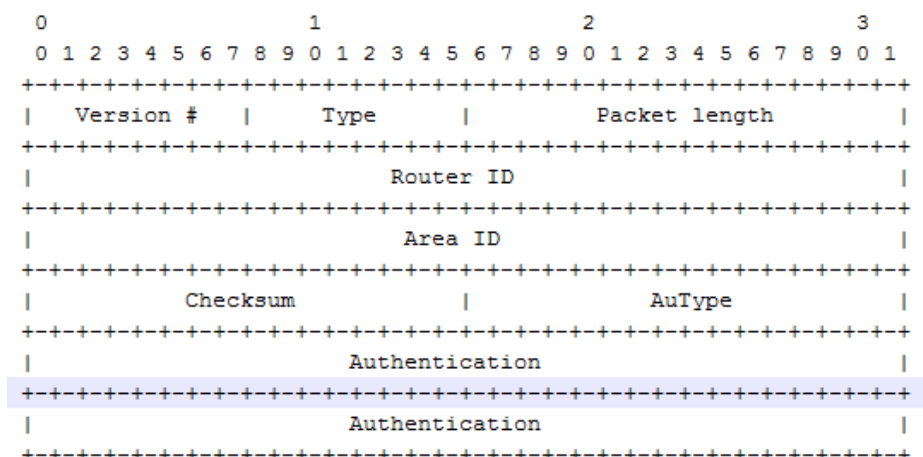
[1]+ Done /etc/init.d/zebra start
r3:~#
```

```
r4:~# /etc/init.d/zebra start &
[1] 580
r4:~# Loading capability module if not yet done.
Starting Quagga daemons (prio:10): zebra ospfd.

[1]+ Done /etc/init.d/zebra start
r4:~#
```

### d) Análisis del archivo ospfd.startup.r1.cap

Hay cinco tipos diferentes de paquetes OSPF. Cada paquete OSPF comienza con una cabecera estándar de 24 bytes. Este encabezamiento contiene toda la información necesaria para determinar si el paquete debe ser aceptado para su posterior procesamiento.



Donde:

- Versión: El número de versión de OSPF.
- Type: Los tipos de paquetes OSPF son:
  - 1 Hello
  - 2 Database Descriptions
  - 3 Link State Request
  - 4 Link State Update
  - 5 Link State Acknowledgment
- Packet length: La longitud del paquete de protocolo OSPF en bytes. Esta longitud incluye la cabecera OSPF estándar.
- Router ID: El ID del router de origen del paquete.
- Area ID: Un número de 32 bits que identifica el área a la cual pertenece el paquete. Todos los paquetes OSPF se asocian con una sola área. La mayoría de los viajes son de un solo salto. Los paquetes que viajan sobre un enlace virtual se etiquetan con el ID de área Backbone 0.0.0.0.
- Checksum: La suma de comprobación IP estándar de todo el contenido del paquete se considera a partir de la cabecera del paquete OSPF pero excluyendo los 64-bit del campo de autenticación. Esta suma de control se calcula como el 16-bit complemento a uno de la suma de los complementos a uno de todas los 16-bit de palabras en el paquete, exceptuando el campo de autenticación. Si la longitud del paquete no es un número entero de palabras de 16-bit, el paquete se completa con un byte nulo. La suma de control se considera como parte del procedimiento de autenticación de paquetes, para algunos tipos de autenticación la suma de comprobación se omite.
- AuType: Identifica el procedimiento de autenticación a usar para el paquete.
- Authentication: Un campo de 64-bit para uso del esquema de autenticación.

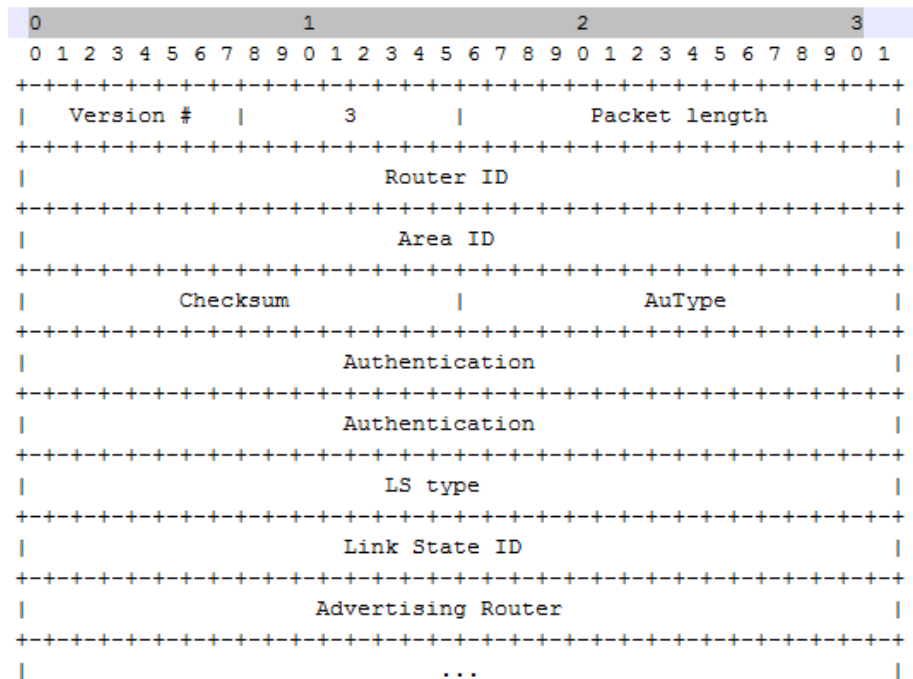
## 1. Características de los mensajes LS Request, LS Update y LS Acknowledge.

- LS Request (Link State Request)

Los paquetes Link State Request son del tipo 3 de paquetes OSPF. Después de intercambiar los paquetes de descripción de base con un router vecino, un router puede encontrar que algunas partes de su base de datos link-state están fuera de fecha. Los paquetes Link State Request se utilizan para solicitar los registros de la base de datos de los vecinos que están más actualizados. Puede que sea necesario el envío de múltiples paquetes Link State.

Un router que envía un paquete Link State Request tiene en consideración los registros de la base de datos que este está solicitando. Cada instancia se define por su número de secuencia LS, LS Checksum, y la LS age, aunque estos campos no son especificados en el Paquete Link State Request en sí. El router puede recibir registros más actualizados en respuesta.

Estructura:



## Captura de Wireshark

63	41.700024	100.1.0.2	100.1.0.1	OSPF	DB Descr.
64	41.700938	100.1.0.1	100.1.0.2	OSPF	DB Descr.
65	41.701302	100.1.0.1	100.1.0.2	OSPF	LS Request
66	41.702838	100.1.0.2	224.0.0.5	OSPF	LS Update
67	41.705426	100.1.0.2	100.1.0.1	OSPF	LS Request
68	41.706407	100.1.0.1	224.0.0.6	OSPF	LS Update
69	41.708928	100.1.0.2	224.0.0.5	OSPF	LS Update

Linux cooked capture
Internet Protocol, Src: 100.1.0.1 (100.1.0.1), Dst: 100.1.0.2 (100.1.0.2)
Open Shortest Path First
OSPF Header
OSPF Version: 2
Message Type: LS Request (3)
Packet Length: 36
Source OSPF Router: 100.1.1.1 (100.1.1.1)
Area ID: 0.0.0.0 (Backbone)
Packet Checksum: 0xccd0 [correct]
Auth Type: Null
Auth Data (none)
Link State Request
Link-State Advertisement Type: Router-LSA (1)
Link State ID: 100.1.2.1
Advertising Router: 100.1.2.1 (100.1.2.1)

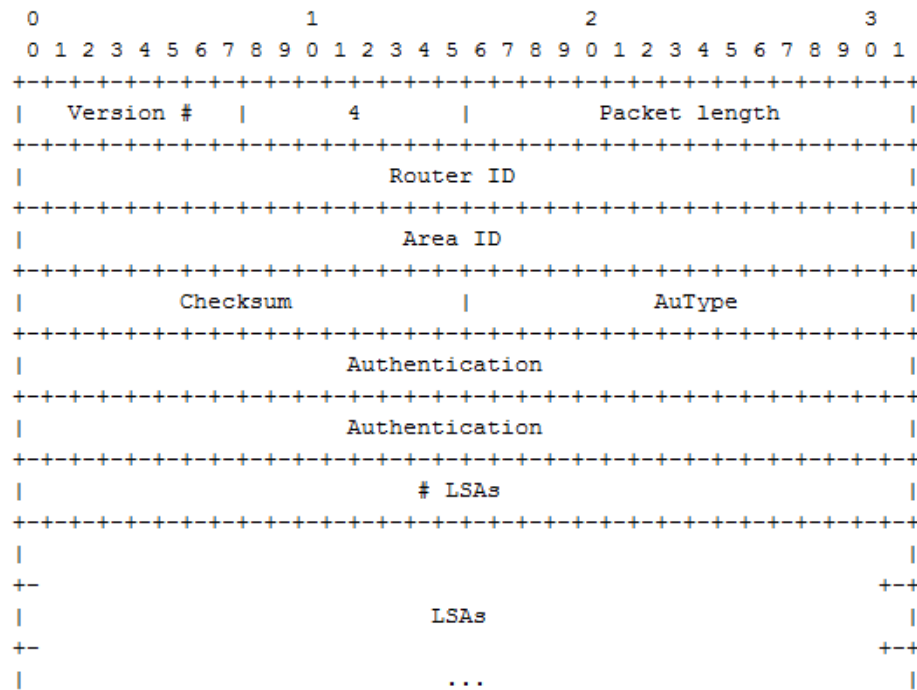
- LSUpdate (Link State Update)

Los paquetes Link State Update son los de tipo 4 dentro de los paquetes OSPF. Estos paquetes implementan la propagación de los LSA's. Cada paquete Link State Update lleva una colección de LSAs un salto más allá de su origen.

Los paquetes Link State Update son multicast en esas redes físicas que soporten multicast / broadcast. Con el fin de que el proceso de propagación sea fiable, los LSAs propagados son reconocidos en los paquetes Link State Acknowledgment. Si la retransmisión de ciertas LSAs es necesaria, los LSAs retransmitidos siempre se envían directamente al vecino.



Estructura:



Donde #LSA's es el número de LSA's enviados en el paquete.

Captura de Whireshark:

79	43.066164	100.1.0.10	100.1.0.9	OSPF	DB Descr.
77	42.701678	100.1.0.2	224.0.0.5	OSPF	LS Acknowledge
76	42.017455	100.1.0.1	224.0.0.5	OSPF	LS Acknowledge
74	41.751350	100.1.0.2	224.0.0.5	OSPF	Hello Packet
72	41.712273	100.1.0.2	224.0.0.5	OSPF	LS Update
71	41.710390	100.1.0.1	100.1.0.2	OSPF	LS Acknowledge
70	41.710011	100.1.0.1	224.0.0.5	OSPF	Hello Packet

Frame 72 (96 bytes on wire, 96 bytes captured)

Linux cooked capture

Internet Protocol, Src: 100.1.0.2 (100.1.0.2), Dst: 224.0.0.5 (224.0.0.5)

Open Shortest Path First

OSPF Header

- OSPF Version: 2
- Message Type: LS Update (4)
- Packet Length: 60
- Source OSPF Router: 100.1.2.1 (100.1.2.1)
- Area ID: 0.0.0.0 (Backbone)
- Packet Checksum: 0x8fc3 [correct]
- Auth Type: Null
- Auth Data (none)

LS Update Packet

- Number of LSAs: 1
- LS Type: Network-LSA

Los paquetes Link State Acknowledgment son el tipo 5 de paquetes OSPF. Para hacer que la propagación de los LSA's sea confiable, los LSAs propagados están explícitamente reconocidos. Este reconocimiento se realiza mediante el envío y la recepción de paquetes Link State Acknowledgment. Múltiples LSAs pueden ser reconocidos en un mismo paquete de este tipo.

El formato de este paquete es similar a la del paquete Data Description. El cuerpo de ambos paquetes es simplemente una lista de encabezados de LSA.

[illegible]

**Página**  
**26**

Captura Whireshark:

72 41.712273	100.1.0.2	224.0.0.5	OSPF	LS Update
71 41.710390	100.1.0.1	100.1.0.2	OSPF	LS Acknowledge
III				
Frame 71 (80 bytes on wire, 80 bytes captured)				
Linux cooked capture				
Internet Protocol, Src: 100.1.0.1 (100.1.0.1), Dst: 100.1.0.2 (100.1.0.2)				
Open Shortest Path First				
OSPF Header				
OSPF Version: 2				
Message Type: LS Acknowledge (5)				
Packet Length: 44				
Source OSPF Router: 100.1.1.1 (100.1.1.1)				
Area ID: 0.0.0.0 (Backbone)				
Packet Checksum: 0x20a0 [correct]				
Auth Type: Null				
Auth Data (none)				
LSA Header				
LS Age: 1 seconds				
Do Not Age: False				
Options: 0x02 (E)				
Link-State Advertisement Type: Router-LSA (1)				
Link State ID: 100.1.2.1				
Advertising Router: 100.1.2.1 (100.1.2.1)				
LS Sequence Number: 0x80000002				
LS Checksum: 0x29e7				
Length: 60				

## 2. ¿Cuál es la dirección de destino de los mensajes Hello? Por qué?

Los paquetes Hello son enviados desde los routers en funcionamiento. Se utilizan para descubrir y mantener relaciones de vecindad. En redes broadcast y NBMA, los paquetes Hello se utilizan también para elegir el Router Designado (DR) y Router designado de respaldo (DR Backup).

En las redes broadcast y redes physical point-to-point, los paquetes Hello se envían periódicamente (cada HelloInterval segundos) a la dirección IP multicast AllSPFRouters. Esta dirección multicast tiene asignado el valor 224.0.0.5 y todos los routers que corran OSPF debe estar preparados para recibir los paquetes enviados a esta dirección. Los paquetes Hello siempre se envían a este destino. Además, ciertos paquetes del protocolo OSPF se envían a esta dirección durante el proceso de propagación.

En otros tipos de redes cambia la forma de propagación de los paquetes Hello, por ejemplo, en los virtual links, los paquetes Hello se envían como unicast (directamente al otro extremo del enlace virtual cada segundo HelloInterval). En Point-to-MultiPoint networks, distintos paquetes Hello se envían a cada vecino adyacente cada segundo HelloInterval.

Captura de Wireshark:

No.	Time	Source	Destination	Protocol	Info
202	83.119154	100.1.0.9	224.0.0.5	OSPF	Hello Packet
201	81.790645	100.1.0.2	224.0.0.5	OSPF	Hello Packet
200	81.747585	100.1.0.1	224.0.0.5	OSPF	Hello Packet
199	80.047782	100.1.1.1	224.0.0.5	OSPF	Hello Packet
198	74.788748	100.1.0.14	224.0.0.5	OSPF	Hello Packet
197	74.770358	100.1.0.13	224.0.0.5	OSPF	Hello Packet
196	73.139488	100.1.0.10	224.0.0.5	OSPF	Hello Packet
195	73.107715	100.1.0.9	224.0.0.5	OSPF	Hello Packet
194	71.781113	100.1.0.2	224.0.0.5	OSPF	Hello Packet
193	71.738182	100.1.0.1	224.0.0.5	OSPF	Hello Packet
192	70.037918	100.1.1.1	224.0.0.5	OSPF	Hello Packet
191	64.786545	100.1.0.14	224.0.0.5	OSPF	Hello Packet
190	64.757671	100.1.0.13	224.0.0.5	OSPF	Hello Packet
189	63.132216	100.1.0.10	224.0.0.5	OSPF	Hello Packet
188	63.097905	100.1.0.9	224.0.0.5	OSPF	Hello Packet
187	61.770925	100.1.0.2	224.0.0.5	OSPF	Hello Packet

Se puede notar que todas las direcciones de destino, para los paquetes Hello, coinciden con la AllSPFRouters address.

### 3. ¿Cuáles son los Router ID para r1...r4?

El Router ID es un número de 32 bits que identifica de forma única el router en el AS.

Según se indica en el RFC un criterio posible sería asignar el router id con la ip menor de entre las IP's asignadas a las interfaces del router. También podría ser tomado considerando la mayor de esas IP's como se muestra a continuación.

El proceso de selección de Router ID consiste en (Cisco):

- Utilizar la dirección IP configurada con el comando router-id de OSPF.
- Si router-id no está configurado, el router elige la dirección IP más alta de cualquiera de sus interfaces loopback.
- Si no hay ninguna loopback configurada, el router elige la dirección IP activa más alta de cualquiera de sus interfaces físicas.

Los Routers IDs quedaron:

- R1 – 100.1.1.1
- R2 – 100.1.2.1
- R3 – 100.1.3.1
- R4 – 100.1.0.14

Si evaluamos los Router IDs designados vemos que el correspondiente a r4 no cumple con el criterio ya que no se trata de la IP mayor de entre las IPs asignadas a sus interfaces (el resto de las designaciones si lo cumple). Investigando se pudo ver que esto se debe al criterio que toma el software “Quagga” que utilizamos en el curso. El mismo permite que se asigne un router-id en el fichero ospfd.conf, y si no es asignado, elige como router-id la IP de la interfaz eth mayor de las todas las interfaces del router (es decir, si el router tiene IPs asignada a eth0, eth1 y eth2, el router-id sería la IP de eth2).

### 4. Análisis del log del ospfd en r1 y comentario sobre el proceso DR-Election.

¿Cuál es el rol del DR y el Backup-DR?

```
r1:/var/log/zebra# cat ospfd.log
2012/11/11 10:34:31 OSPF: OSPFd 0.99.10 starting: vty@2604
2012/11/11 10:34:31 OSPF: interface 100.1.0.13 [3] join AllSPFRouters Multicast group.
2012/11/11 10:34:31 OSPF: interface 100.1.0.9 [4] join AllSPFRouters Multicast group.
2012/11/11 10:34:31 OSPF: interface 100.1.0.1 [5] join AllSPFRouters Multicast group.
2012/11/11 10:34:31 OSPF: interface 100.1.1.1 [6] join AllSPFRouters Multicast group.
2012/11/11 10:34:33 OSPF: ospfTrapNbrStateChange trap sent: 100.1.0.2 now Init/DROther
2012/11/11 10:34:34 OSPF: ospfTrapNbrStateChange trap sent: 100.1.0.10 now Init/DROther
2012/11/11 10:34:36 OSPF: ospfTrapNbrStateChange trap sent: 100.1.0.14 now Init/DROther
2012/11/11 10:35:11 OSPF: DR-Election[1st]: Backup 100.1.0.13
2012/11/11 10:35:11 OSPF: DR-Election[1st]: DR 100.1.0.13
2012/11/11 10:35:11 OSPF: DR-Election[2nd]: Backup 100.1.0.14
2012/11/11 10:35:11 OSPF: DR-Election[2nd]: DR 100.1.0.13
2012/11/11 10:35:11 OSPF: ospfTrapIfStateChange trap sent: 100.1.0.13 now DR
2012/11/11 10:35:11 OSPF: interface 100.1.0.13 [3] join AllDRouters Multicast group.
2012/11/11 10:35:11 OSPF: DR-Election[1st]: Backup 100.1.0.10
2012/11/11 10:35:11 OSPF: DR-Election[1st]: DR 100.1.0.10
2012/11/11 10:35:11 OSPF: ospfTrapIfStateChange trap sent: 100.1.0.9 now DROther
2012/11/11 10:35:11 OSPF: DR-Election[1st]: Backup 100.1.0.2
2012/11/11 10:35:11 OSPF: DR-Election[1st]: DR 100.1.0.2
2012/11/11 10:35:11 OSPF: ospfTrapIfStateChange trap sent: 100.1.0.1 now DROther
2012/11/11 10:35:11 OSPF: DR-Election[1st]: Backup 100.1.1.1
2012/11/11 10:35:11 OSPF: DR-Election[1st]: DR 100.1.1.1
2012/11/11 10:35:11 OSPF: DR-Election[2nd]: Backup 0.0.0.0
2012/11/11 10:35:11 OSPF: DR-Election[2nd]: DR 100.1.1.1
2012/11/11 10:35:11 OSPF: ospfTrapIfStateChange trap sent: 100.1.1.1 now DR
2012/11/11 10:35:11 OSPF: interface 100.1.1.1 [6] join AllDRouters Multicast group.
2012/11/11 10:35:13 OSPF: Packet[DD]: Neighbor 100.1.2.1 Negotiation done (Slave).
2012/11/11 10:35:13 OSPF: nsm_change_state(100.1.2.1, Loading -> Full): scheduling new router-LSA origination
2012/11/11 10:35:13 OSPF: DR-Election[1st]: Backup 100.1.0.1
2012/11/11 10:35:13 OSPF: DR-Election[1st]: DR 100.1.0.2
2012/11/11 10:35:13 OSPF: DR-Election[2nd]: Backup 100.1.0.1
2012/11/11 10:35:13 OSPF: DR-Election[2nd]: DR 100.1.0.2
2012/11/11 10:35:13 OSPF: ospfTrapIfStateChange trap sent: 100.1.0.1 now Backup
2012/11/11 10:35:13 OSPF: interface 100.1.0.1 [5] join AllDRouters Multicast group.
2012/11/11 10:35:14 OSPF: Packet[DD]: Neighbor 100.1.3.1 Negotiation done (Slave).
2012/11/11 10:35:14 OSPF: nsm_change_state(100.1.3.1, Exchange -> Full): scheduling new router-LSA origination
2012/11/11 10:35:14 OSPF: DR-Election[1st]: Backup 100.1.0.9
2012/11/11 10:35:14 OSPF: DR-Election[1st]: DR 100.1.0.10
2012/11/11 10:35:14 OSPF: DR-Election[2nd]: Backup 100.1.0.9
2012/11/11 10:35:14 OSPF: DR-Election[2nd]: DR 100.1.0.10
2012/11/11 10:35:14 OSPF: ospfTrapIfStateChange trap sent: 100.1.0.9 now Backup
2012/11/11 10:35:14 OSPF: interface 100.1.0.9 [4] join AllDRouters Multicast group.
2012/11/11 10:35:16 OSPF: Packet[DD]: Neighbor 100.1.0.14: Initial DBD from Slave, ignoring.
2012/11/11 10:35:16 OSPF: DR-Election[1st]: Backup 100.1.0.14
2012/11/11 10:35:16 OSPF: DR-Election[1st]: DR 100.1.0.13
2012/11/11 10:35:16 OSPF: Packet[DD]: Neighbor 100.1.0.14 Negotiation done (Master).
2012/11/11 10:35:16 OSPF: ospfTrapNbrStateChange trap sent: 100.1.0.14 now Loading/Backup
2012/11/11 10:35:16 OSPF: nsm_change_state(100.1.0.14, Loading -> Full): scheduling new router-LSA origination
r1:/var/log/zebra#
```

### Elección de Designated Router (DR) y Backup Designated Router (BDR)

El algoritmo utilizado para calcular el Designated Router y el Backup Designated Router de la red se invoca por la Interface state machine. En el momento en que un router ejecuta el algoritmo de elección para una red, el DR y BDR se inicializan en 0.0.0.0. Esto indica la falta tanto de un DR y BDR.

El algoritmo de elección es el siguiente: Llamamos al router que realiza la elección "Router X". Se considera la lista de routers a examinar, la cual se conforma con los routers vecinos, siempre que se haya establecido conexión bidireccional con el Router X. Esta lista de vecinos del Router X se corresponde a la lista de vecinos que poseen estado mayor o igual que 2-Way, dentro de dicha colección se incluye al propio Router X. Además se deben de descartar de dicha lista a los routers que no son elegibles como DR, es decir los que poseen prioridad 0. Los pasos que se detallan a continuación solo consideran los Routers incluidos en dicha lista.

1- Se toman en cuenta los valores actuales para DR y BDR de la red.

2- Se calcula el nuevo BDR de la red de la siguiente forma. Solo los routers de la lista que no se hayan declarado a sí mismos como DR son elegibles para pasar a ser BDR. Si uno o más routers se declaran como BDR (es decir que se muestran como BDR, pero no DR, en sus paquetes Hello) se elige como BDR al que tenga mayor prioridad asignada. En caso de empate, el que tenga el mayor Router ID es elegido. Si ningún router se ha declarado a sí mismo como BDR, se elige el router con mayor prioridad asignada de entre los routers que no se han declarado DR, en caso de empate se designa al que tiene mayor router ID.

3- Se calcula el DR para la red de la siguiente forma. Si uno o más routers de declararon a sí mismos como DR (es decir, que se muestran a sí mismos como DR en sus paquetes Hello) se elige como DR al que posee la mayor prioridad asignada. En caso de empate, el router con el mayor router ID es elegido. Si ningún router se declaró a sí mismo como DR, es elegido el más reciente BDR para que también sea DR.

4- Si el Router X es ahora el nuevo DR o BDR, o si dejó de ser DR o BDR, se repiten los pasos 2 y 3, y luego se procede a realizar el paso 5. Por ejemplo, si el Router X pasó a ser DR, cuando el paso 2 se repita, el Router X ya no será elegible como BDR. Una de las cosas que esto asegura es que ningún router pueda declararse DR y BDR simultáneamente.

5- Como resultado de estos pasos, el router podrá ser ahora el DR o el BDR. La interfaz del router deberá ser asignada en correspondencia. Si el router paso a ser BDR, la interfaz será Backup, si paso a ser DR la interfaz será DR, en otro caso se asignara DR other.

6- Si la red en cuestión se trata de una red NBMA, y el router paso a ser DR o bien BDR, este debe iniciar el envío de paquetes Hello a sus vecinos que no sean elegibles como DR. Esto se realiza mediante la invocación del evento Start en cada vecino que tenga prioridad 0.

7- Si este procedimiento causa el cambio en la asignación del DR o BDR, las adyacencias asociadas con su interfaz necesitaran ser modificadas. Algunas necesitaran ser generadas, otras eliminadas. Para realizar esto, se invoca al evento AdjOK? en todos los vecinos cuyo estado sea al menos 2-Way, lo que causara que su adyacencia sea reexaminada.

La razón de la complejidad del algoritmo de elección es el deseo de una transición ordenada de BDR a DR cuando el DR actual falla. Esta transición está asegurada a través de la introducción de la histéresis: ningún BDR se puede elegir hasta que el antiguo BDR acepta sus nuevas responsabilidades de DR.

Con el procedimiento anterior se puede elegir el mismo router como DR y BDR, sin embargo en caso de cumplirse esto nunca se tratará del router X. El DR elegido no necesariamente es el que tiene la prioridad más alta, ni el BDR necesariamente tiene la segunda prioridad más alta. Si el Router X no es elegible para ser DR, es posible que ni BDR ni el DR sean seleccionados en el procedimiento en cuestión. Se debe tener en cuenta que si Router X es el único router conectado que es elegible para convertirse en DR, este se seleccionará a sí mismo como DR y no habrá BDR para la red.

### Roles:

Cada red multiaccess tiene un DR. Este es el puesto que toma uno de los routers de cada segmento multiacceso. La elección del DR tiene como origen el problema que representan las múltiples adyacencias y saturación extensa de LSA. Es decir que el DR toma el papel de representar el punto de recolección y distribución de los LSAs enviados y recibidos.

El BDR es otro router que realiza intercambios con los demás nodos del segmento multiacceso. Se utiliza para agregar redundancia, ya que si el DR se cae, el BDR lo reemplazará para mantener la convergencia.

El resto de los routers (los que no son DR ni BDR) se denominan DROthers. Estos solo forman adyacencias completas y envían sus LSA solo con el DR y BDR de la red. La forma en que se envían estos LSA al DR es asignando la dirección de destino multicast 200.0.0.6 (ALLDRouters – All DR Routers), y de forma inversa el DR utiliza la dirección multicast 200.0.0.5 (AllSPDRouters – All OSPF Routers).

## e) Continuación del laboratorio

f) ¿Cómo se redistribuye la ruta por defecto hacia r5 en OSPF? Verifique que se propague efectivamente en r1...r3.

Estado de las tablas de ruteo de r1, r2 y r3 antes de propagar la ruta por defecto:

r1							
r1:~# route							
Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
100.1.0.16	100.1.0.14	255.255.255.252	UG	20	0	0	eth0
100.1.0.0	*	255.255.255.252	U	0	0	0	eth2
100.2.0.0	100.1.0.14	255.255.255.252	UG	20	0	0	eth0
100.1.0.4	100.1.0.10	255.255.255.252	UG	20	0	0	eth1
100.1.0.8	*	255.255.255.252	U	0	0	0	eth1
100.1.0.12	*	255.255.255.252	U	0	0	0	eth0
100.1.4.0	100.1.0.14	255.255.255.0	UG	20	0	0	eth0
100.1.2.0	100.1.0.2	255.255.255.0	UG	20	0	0	eth2
100.1.3.0	100.1.0.10	255.255.255.0	UG	20	0	0	eth1
100.1.1.0	*	255.255.255.0	U	0	0	0	eth3
r1:~#							
r2							
r2:~# route							
Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
100.1.0.16	100.1.0.6	255.255.255.252	UG	20	0	0	eth1
100.1.0.0	*	255.255.255.252	U	0	0	0	eth0
100.2.0.0	100.1.0.1	255.255.255.252	UG	20	0	0	eth0
100.1.0.4	*	255.255.255.252	U	0	0	0	eth1
100.1.0.8	100.1.0.1	255.255.255.252	UG	20	0	0	eth0
100.1.0.12	100.1.0.1	255.255.255.252	UG	20	0	0	eth0
100.1.4.0	100.1.0.1	255.255.255.0	UG	30	0	0	eth0
100.1.2.0	*	255.255.255.0	U	0	0	0	eth2
100.1.3.0	100.1.0.6	255.255.255.0	UG	20	0	0	eth1
100.1.1.0	100.1.0.1	255.255.255.0	UG	20	0	0	eth0
r2:~#							
r3							
r3:~# route							
Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
100.1.0.16	*	255.255.255.252	U	0	0	0	eth2
100.1.0.0	100.1.0.5	255.255.255.252	UG	20	0	0	eth0
100.2.0.0	100.1.0.18	255.255.255.252	UG	20	0	0	eth2
100.1.0.4	*	255.255.255.252	U	0	0	0	eth0
100.1.0.8	*	255.255.255.252	U	0	0	0	eth1
100.1.0.12	100.1.0.9	255.255.255.252	UG	20	0	0	eth1
100.1.4.0	100.1.0.18	255.255.255.0	UG	20	0	0	eth2
100.1.2.0	100.1.0.5	255.255.255.0	UG	20	0	0	eth0
100.1.3.0	*	255.255.255.0	U	0	0	0	eth3
100.1.1.0	100.1.0.9	255.255.255.0	UG	20	0	0	eth1
r3:~#							



Para propagar la ruta por defecto:

```
▲r4:~# telnet localhost ospfd
Trying 127.0.0.1...
Connected to r4.
Escape character is '^]'.

Hello, this is Quagga (version 0.99.10).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

Password:
ospfd> enable
Password:
ospfd# configure terminal
ospfd(config)# router ospd
% [OSPF] Unknown command: router ospd
ospfd(config)# router ospf
ospfd(config-router)# redistribute static
ospfd(config-router)# default-information originate
ospfd(config-router)# write
Configuration saved to /etc/quagga/ospfd.conf
ospfd(config-router)# quit
ospfd(config)# quit
ospfd# exit
Connection closed by foreign host.
▼r4:~#
```

Conceptos utilizados:

Ruta Estática: Son rutas indicadas al router manualmente por un administrador, de forma que se pueda generar el enrutamiento al destino deseado. Esto ocurre a diferencia del caso de las rutas dinámicas, las cuales son indicadas automáticamente al router mediante protocolos de ruteo.

Ruta por Defecto: Son rutas que surgen ante la necesidad de generar envío de tráfico a rutas que no se encuentran en las correspondientes tablas de ruteo de los dispositivos de la red. Un ejemplo claro de redes donde son utilizadas son las redes con acceso a internet.

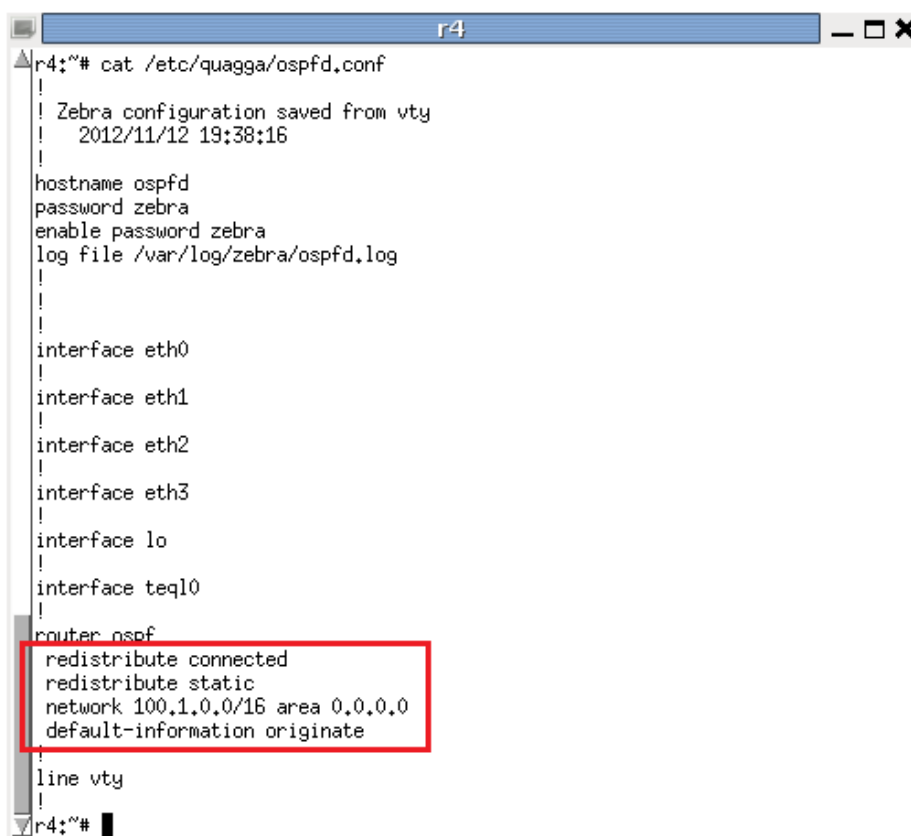
Configuración:

*Redistribute static* es utilizado de forma de propagar una ruta estática en una red OSPF. Es decir que el router se encargara de distribuir la información sobre la ruta estática que se encuentra disponible y se añadió manualmente.

*Default-information originate* se utiliza para propagar una ruta por defecto en OSPF.

A pesar del comando *redistribute static* ya comentado, no es posible en OSPF crear una ruta por defecto solamente redistribuyendo una ruta estática. Incluso habiendo una ruta por defecto presente en la tabla de ruteo, por defecto no se forwardeara al resto de la red. Esto se debe a que OSPF utiliza un algoritmo de link-state que realiza un seguimiento de los links en lugar de las rutas. Dado todo esto hay que indicar explícitamente de alguna forma que se quiere redistribuir la ruta por defecto mediante el comando *default-information originate*.

El archivo de configuración de ospf, ospfd.conf, queda:



```
r4
r4:~# cat /etc/quagga/ospfd.conf
!
! Zebra configuration saved from vty
!   2012/11/12 19:38:16
!
hostname ospfd
password zebra
enable password zebra
log file /var/log/zebra/ospfd.log
!
!
interface eth0
!
interface eth1
!
interface eth2
!
interface eth3
!
interface lo
!
interface teql0
!
router ospf
  redistribute connected
  redistribute static
  network 100.1.0.0/16 area 0.0.0.0
  default-information originate
!
line vty
!
r4:~#
```

Estado de las tablas de ruteo de r1, r2 y r3 luego de agregar la ruta por defecto:

**r1**

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
100.1.0.16	100.1.0.14	255.255.255.252	UG	20	0	0	eth0
100.1.0.0	*	255.255.255.252	U	0	0	0	eth2
100.1.0.4	100.1.0.10	255.255.255.252	UG	20	0	0	eth1
100.2.0.0	100.1.0.14	255.255.255.252	UG	20	0	0	eth0
100.1.0.8	*	255.255.255.252	U	0	0	0	eth1
100.1.0.12	*	255.255.255.252	U	0	0	0	eth0
100.1.4.0	100.1.0.14	255.255.255.0	UG	20	0	0	eth0
100.1.2.0	100.1.0.2	255.255.255.0	UG	20	0	0	eth2
100.1.3.0	100.1.0.10	255.255.255.0	UG	20	0	0	eth1
100.1.1.0	*	255.255.255.0	U	0	0	0	eth3
default	100.1.0.14	0.0.0.0	UG	10	0	0	eth0

**r2**

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
100.1.0.16	100.1.0.6	255.255.255.252	UG	20	0	0	eth1
100.1.0.0	*	255.255.255.252	U	0	0	0	eth0
100.2.0.0	100.1.0.1	255.255.255.252	UG	20	0	0	eth0
100.1.0.4	*	255.255.255.252	U	0	0	0	eth1
100.1.0.8	100.1.0.1	255.255.255.252	UG	20	0	0	eth0
100.1.0.12	100.1.0.1	255.255.255.252	UG	20	0	0	eth0
100.1.4.0	100.1.0.1	255.255.255.0	UG	30	0	0	eth0
100.1.2.0	*	255.255.255.0	U	0	0	0	eth2
100.1.3.0	100.1.0.6	255.255.255.0	UG	20	0	0	eth1
100.1.1.0	100.1.0.1	255.255.255.0	UG	20	0	0	eth0
default	100.1.0.1	0.0.0.0	UG	10	0	0	eth0

**r3**

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
100.1.0.16	*	255.255.255.252	U	0	0	0	eth2
100.1.0.0	100.1.0.5	255.255.255.252	UG	20	0	0	eth0
100.2.0.0	100.1.0.18	255.255.255.252	UG	20	0	0	eth2
100.1.0.4	*	255.255.255.252	U	0	0	0	eth0
100.1.0.8	*	255.255.255.252	U	0	0	0	eth1
100.1.0.12	100.1.0.9	255.255.255.252	UG	20	0	0	eth1
100.1.4.0	100.1.0.18	255.255.255.0	UG	20	0	0	eth2
100.1.2.0	100.1.0.5	255.255.255.0	UG	20	0	0	eth0
100.1.3.0	*	255.255.255.0	U	0	0	0	eth3
100.1.1.0	100.1.0.9	255.255.255.0	UG	20	0	0	eth1
default	100.1.0.18	0.0.0.0	UG	10	0	0	eth2

Se puede ver que la ruta efectivamente fue propagada al resto de la red.

## g) Conexión al demonio ospf en r3 e identificación de las rutas externas de OSPF tipo 2 (E2)

```

r3
ospfd> sh ip ospf route
===== OSPF network routing table =====
N 100.1.0.0/30      [20] area: 0.0.0.0
                        via 100.1.0.5, eth0
                        via 100.1.0.9, eth1
N 100.1.0.4/30      [10] area: 0.0.0.0
                        directly attached to eth0
N 100.1.0.8/30      [10] area: 0.0.0.0
                        directly attached to eth1
N 100.1.0.12/30     [20] area: 0.0.0.0
                        via 100.1.0.9, eth1
                        via 100.1.0.18, eth2
N 100.1.0.16/30     [10] area: 0.0.0.0
                        directly attached to eth2
N 100.1.1.0/24      [20] area: 0.0.0.0
                        via 100.1.0.9, eth1
N 100.1.2.0/24      [20] area: 0.0.0.0
                        via 100.1.0.5, eth0
N 100.1.3.0/24      [10] area: 0.0.0.0
                        directly attached to eth3
N 100.1.4.0/24      [20] area: 0.0.0.0
                        via 100.1.0.18, eth2

===== OSPF router routing table =====
R 100.1.0.14        [10] area: 0.0.0.0, ASBR
                        via 100.1.0.18, eth2
R 100.1.1.1         [10] area: 0.0.0.0, ASBR
                        via 100.1.0.9, eth1
R 100.1.2.1         [10] area: 0.0.0.0, ASBR
                        via 100.1.0.5, eth0

===== OSPF external routing table =====
N E2 0.0.0.0/0      [10/10] tag: 0
                        via 100.1.0.18, eth2
N E2 100.2.0.0/30   [10/20] tag: 0
                        via 100.1.0.18, eth2

ospfd>
Vty connection is timed out.
Connection closed by foreign host.
r3:~#
  
```

OSPF utiliza métricas de enrutamiento basadas en el path cost. Se definen varios tipos de métricas y se debe tener en cuenta que estas pueden ser comparadas siempre y cuando pertenezcan al mismo tipo. Existen métricas para rutas externas e internas. Tenemos que una ruta intra-zona es siempre preferible a una ruta externa independientemente de métrica.

Como ejemplo de tipos de rutas externas en OSPF, en relación a su métrica tenemos:

E1 o rutas externas Type1: Su costo es el costo de la métrica externa sumándole el costo adicional de la red interna OSPF necesario para llegar a la red externa.

E2 o rutas externas Type2: Su costo siempre es la métrica externa, no se toma el costo interno de la red OSPF para alcanzar la red externa.

### Comentarios:

Como se muestra en la figura podemos observar dos rutas externas OSPF.

- 0.0.0.0/0 la default route. Dado que se está en un área stub, es decir que no se acepta información de rutas externas al AS, como por ejemplo las provenientes de otros orígenes no OSPF. Lo que se realiza para enrutar hacia afuera del AS es utilizar esta ruta por defecto.
- 100.2.0.0/30 Se tiene que la redistribución de rutas en OSPF, a partir de rutas estáticas, genera que las mismas se conviertan en rutas externas. Dado esto es que aparece esta entrada, que representa las rutas estáticas añadidas a la tabla de ruteo de r4 y luego propagadas por OSPF por lo realizado en la parte f.

Se puede observar que para ambas se indica que son vía 100.1.0.18, es decir que son a través de la interfaz eth2 del router r4.

### En la figura se especifica un costo para cada ruta externa:

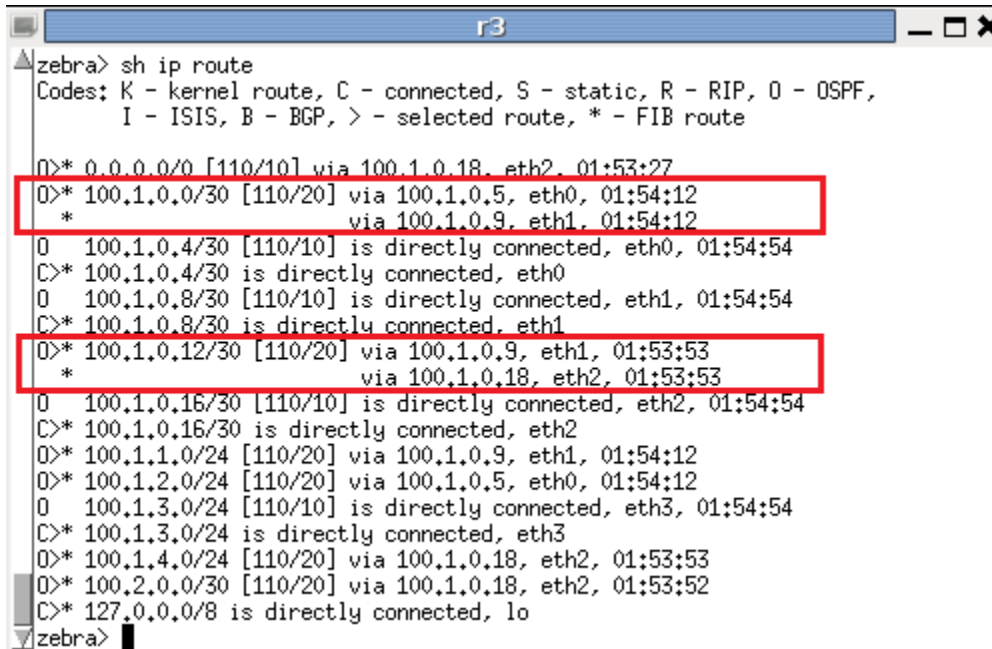
El RFC no especifica un costo por defecto de interfaz para OSPF, solamente que el mismo debe de ser mayor a cero. Eso deja la posibilidad de que las implementaciones varíen en dicho valor. Igualmente se toman valores comunes copiándolos de los que toman grandes proveedores como Cisco Systems.

Cisco utiliza una métrica  $10^8/\text{bandwidth}$  (es el valor base,  $10^8$  por defecto, el mismo puede ser ajustado). Así, un enlace de 100 Mbit / s tendrá un costo de 1, ***uno de 10Mbit/s tendrá costo de 10 (pensamos que el costo de la default route de 10, indicado en la figura como [10/10], se puede deber a una precondition de esta clase) y así sucesivamente.***

Se debe tomar en cuenta que si en la configuración del router, al momento de redistribuir las rutas (*redistribute static*) no se utiliza la palabra reservada *metric*, o si no se utiliza la configuración *default-metric* en el router, se tendrá que el costo asociado a la ruta externa será de 20 (***como se ve en la figura [10/20, para el caso de la ruta 100.2.0.0/30].***

### h) Conexión al demonio zebra en r3.

Identificación de *Next Hop* para las redes 100.1.0.12/30 y 100.1.0.0/30.  
Comparación con el resultado obtenido con RIP.



```
zebra> sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
      I - ISIS, B - BGP, > - selected route, * - FIB route

O>* 0.0.0.0/0 [110/10] via 100.1.0.18, eth2, 01:53:27
O>* 100.1.0.0/30 [110/20] via 100.1.0.5, eth0, 01:54:12
    *                via 100.1.0.9, eth1, 01:54:12
O  100.1.0.4/30 [110/10] is directly connected, eth0, 01:54:54
C>* 100.1.0.4/30 is directly connected, eth0
O  100.1.0.8/30 [110/10] is directly connected, eth1, 01:54:54
C>* 100.1.0.8/30 is directly connected, eth1
O>* 100.1.0.12/30 [110/20] via 100.1.0.9, eth1, 01:53:53
    *                via 100.1.0.18, eth2, 01:53:53
O  100.1.0.16/30 [110/10] is directly connected, eth2, 01:54:54
C>* 100.1.0.16/30 is directly connected, eth2
O>* 100.1.1.0/24 [110/20] via 100.1.0.9, eth1, 01:54:12
O>* 100.1.2.0/24 [110/20] via 100.1.0.5, eth0, 01:54:12
O  100.1.3.0/24 [110/10] is directly connected, eth3, 01:54:54
C>* 100.1.3.0/24 is directly connected, eth3
O>* 100.1.4.0/24 [110/20] via 100.1.0.18, eth2, 01:53:53
O>* 100.2.0.0/30 [110/20] via 100.1.0.18, eth2, 01:53:52
C>* 127.0.0.0/8 is directly connected, lo
zebra>
```

Un next hop es la interfaz de salida de un router para el reenvío de tráfico a un determinado destino

Podemos ver que OSPF nos brinda los siguientes next hop para las redes indicadas:

- Para 100.1.0.0/30 tenemos:
  - 100.1.0.5 - Hacia r2 por interfaz eth1
  - 100.1.0.9 - Hacia r1 por interfaz eth1
- Para 100.1.0.12/30 tenemos:
  - 100.1.0.9 - Hacia r1 por interfaz eth1
  - 100.1.0.18 - Hacia r4 por interfaz eth2.

La diferencia notada se ve en que OSPF, a diferencia de RIP, puede guardar más un camino de menor costo. Dado esto, tenemos que si existen varios caminos de menor costo hacia un destino, todos ellos serán descubiertos y utilizados. De forma que el router en cuestión tendrá varios next hops disponibles hacia dicho destino.

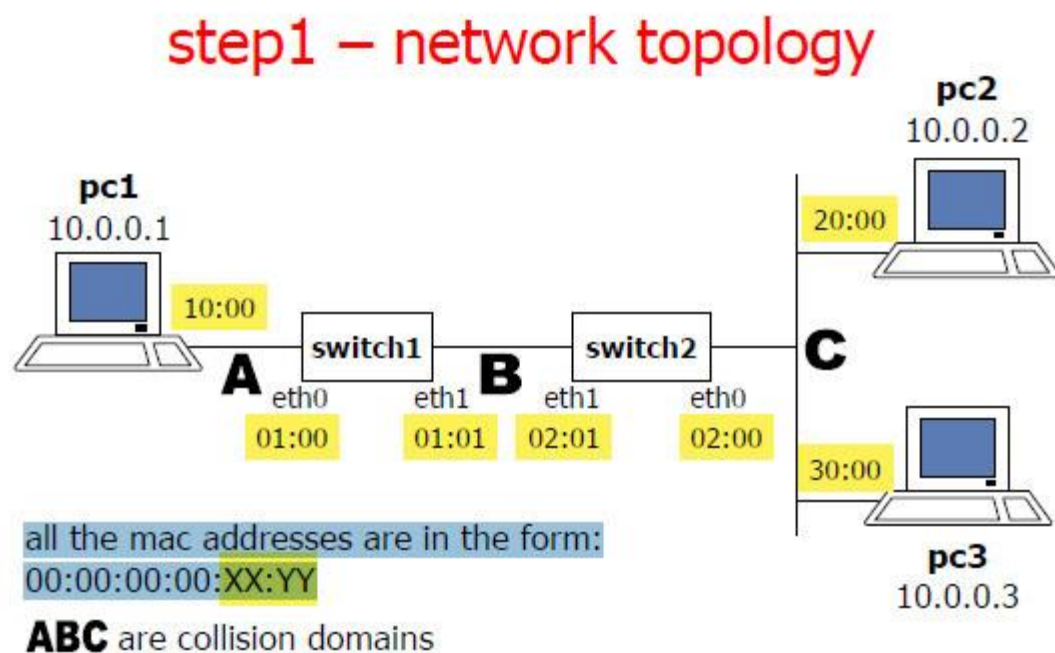
# Laboratorio de Bridging (two-switches)

Se llama Bridging a la acción que permite que dos o más redes (o segmentos de redes) separadas formen una red en común. Con un switch inteligente podemos filtrar los paquetes de una parte de la red para solo transmitir hacia la otra red los necesarios (aquellos dirigidos a la otra red).

## a) Instalación del laboratorio de Bridging.

Para ejecutar se debe entrar al directorio y ejecutar **lstart**. Esto crea varias maquinas virtuales según la topología de la red que representan switches y pcs en este caso.

## b) Vinculación de lab.conf con la topología de red.



pc1[0]="A"

switch1[0]="A"

**pc1 y switch1 están conectados al mismo dominio de colisión "A"**

pc2[0]="C"

pc3[0]="C"

switch2[0]="C"

**pc2 y pc3 se encuentran conectados al switch2 por el dominio de colisión "C"**

switch1[1]="B"

switch2[1]="B"

**switch1 y switch2 están conectados al mismo dominio de colisión "B"**



### c) Verificación del plan de numeración (IP y MAC) con la diapositiva.

Ejecuto el comando **ifconfig** para pc1, pc2 y pc3 analizando la IP (inet addr) y la MAC (HWadd) comprobamos que son iguales. Para Switch1 y Switch2 hago lo mismo verificando la dirección MAC de las interfaces (eth0 y eth1).

Las direcciones MAC son de la forma: 00:00:00:00:XX:YY. Los switch son dispositivos de capa de enlace por lo que no requieren una dirección IP.

Este es el resultado de la ejecución del comando que coincide con la imagen de la diapositiva

	IP (inet addr)	MAC (HWadd)
<b>PC1</b>	10.0.0.1	10:00
<b>PC2</b>	10.0.0.2	20:00
<b>PC3</b>	10.0.0.3	30:00
<b>Switch1</b>		eth0 = 01:00 y eth1 =01:01
<b>Switch2</b>		eth0 = 02:00 y eth1 =02:01

### d) Modificación del plan de numeración IP para pertenecer al prefijo 172.31.0.0/24

Para esto utilizo el comando: **"ifconfig eth0 IP up"** en cada una de las PCs

**Para PC1:** ifconfig eth0 172.31.0.1 up

**Para PC2:** ifconfig eth0 172.31.0.2 up

**Para PC3:** ifconfig eth0 172.31.0.3 up

### e) Modificación de direcciones MAC para emular tarjetas de red de Speakercraft Inc.

Los primeros tres bytes de una dirección MAC es un identificador único organizacional (OUI - Organizationally Unique Identifier) que identifica a la empresa que la fabrica.

La OUI del fabricante Speakercraft es: **00:1F:40** (ref: <http://mac.freenetworks.org/ouis/7946-mac-address-001F40>)

Estos 3 bytes los utilizaremos como prefijo de las direcciones MAC aplicando el comando **"ifconfig eth0 hw ether MAC\_ADDRESS"** para modificar la dirección MAC.

**pc1:** ifconfig eth0 hw ether 00:1F:40:00:10:00

**pc2:** ifconfig eth0 hw ether 00:1F:40:00:20:00

**pc3:** ifconfig eth0 hw ether 00:1F:40:00:30:00

**Switch1 eth0:** ifconfig eth0 hw ether 00:1F:40:00:01:00

**Switch1 eth1:** ifconfig eth1 hw ether 00:1F:40:00:01:01

**Switch2 eth0:** ifconfig eth0 hw ether 00:1F:40:00:02:00

**Switch2 eth1:** ifconfig eth1 hw ether 00:1F:40:00:02:01

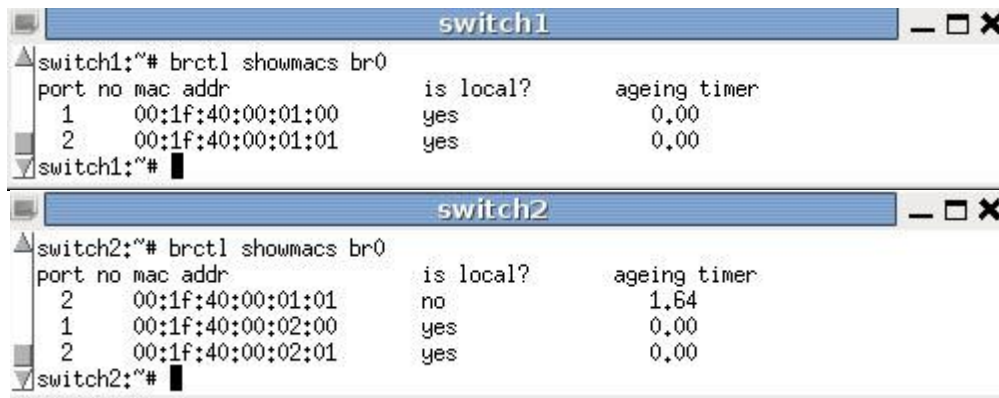
### f) Comandos utilizados para realizar las dos partes anteriores.

Ya fueron comentados en las partes correspondientes.

### g) Análisis de los cambios en el contenido de las tablas de direcciones MAC de los switches de la topología, antes y después de cada comando ejecutado.

1. Genere tráfico de pc2 a pc3 pero previamente y utilizando el comando `tcpdump`, capture el tráfico en pc1 y en pc3 en los archivos `/hosthome/ping.pc1.g1.cap` y `/hosthome/ping.pc3.g1.cap`. Luego de unos 15 segundos de estar generando tráfico, detenga el ping y las capturas. Analice con `wireshark` las capturas realizadas y explique el comportamiento observado.

Para obtener las tablas de direcciones MAC se utilizó el comando `brctl showmacs br0` para ambos switches. Estas son las tablas iniciales para switch1 y switch2:



The image shows two terminal windows. The top window is titled 'switch1' and shows the command 'brctl showmacs br0' being executed. The output is a table with columns 'port', 'no', 'mac', 'addr', 'is local?', and 'ageing timer'. The bottom window is titled 'switch2' and shows the same command being executed. Its output is also a table with the same columns.

port	no	mac	addr	is local?	ageing timer
1	00:1f:40:00:01:00	yes		0.00	
2	00:1f:40:00:01:01	yes		0.00	

port	no	mac	addr	is local?	ageing timer
2	00:1f:40:00:01:01	no		1.64	
1	00:1f:40:00:02:00	yes		0.00	
2	00:1f:40:00:02:01	yes		0.00	

Cuando no se genera tráfico las tablas solamente contienen información de los puertos locales, aunque es posible que alguna maquina genere trafico aun sin ser solicitado, por ejemplo enviando paquetes broadcast. En nuestro caso la primera entrada del switch2 es por paquetes intercambiados para el cálculo del spanning tree.

### Capturando tráfico en pc1 y pc3

**Para pc1:** `tcpdump -i eth0 -w /hosthome/ping.pc1.g1.cap`

**Para pc3:** `tcpdump -i eth0 -w /hosthome/ping.pc3.g1.cap`

**Genero trafico usando el comando ping desde pc2 hacia pc3**

`ping 172.31.0.3` (esto se ejecuta en pc2)

Luego utilizo wireshark para analiza el contenido de los archivos generados.

Para pc1:

No. ↓	Time	Source	Destination	Protocol	Info
1	0.000000	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
2	1.997436	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
3	3.997514	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
4	5.997346	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
5	7.997271	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
6	9.997304	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
7	11.508076	Speaker_00:20:00	Broadcast	ARP	Who has 172.31.0.3? Tell 172.31.0.2
8	11.997405	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
9	13.997531	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
10	15.992165	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
11	17.993170	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
12	19.992012	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
13	21.992164	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
14	23.992100	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
15	25.991980	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
16	27.990794	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
17	29.990810	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00
18	31.990665	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00

Todos los paquetes menos el numero 7 están usando el protocolo Spanning Tree que sirve para gestionar los bucles en la topología de red, este protocolo utiliza paquetes llamados BPDU (Bridge Protocol Data Units). Cuando se dan estos bucles puede pasar que se reenvíen indefinidamente los paquetes broadcast creando un bucle infinito que consumiría mucho ancho de banda pudiendo degradar la red en muy poco tiempo.

El paquete número 7 es un ARP request desde PC2 preguntando quién tiene la IP 172.31.0.3. Este paquete se envía por broadcast a toda la red.

## Para pc3:

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
2	1.510205	Speaker_00:20:00	Broadcast	ARP	Who has 172.31.0.3? Tell 172.31.0.2
3	1.510263	Speaker_00:30:00	Speaker_00:20:00	ARP	172.31.0.3 is at 00:1f:40:00:30:00
4	1.511429	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
5	1.511548	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
6	2.000079	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
7	2.509896	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
8	2.509920	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
9	3.509849	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
10	3.509873	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
11	4.000074	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
12	4.508932	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
13	4.508955	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
14	5.517370	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
15	5.517392	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
16	5.994801	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
17	6.504626	Speaker_00:30:00	Speaker_00:20:00	ARP	Who has 172.31.0.2? Tell 172.31.0.3
18	6.504928	Speaker_00:20:00	Speaker_00:30:00	ARP	172.31.0.2 is at 00:1f:40:00:20:00
19	6.516283	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
20	6.516301	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
21	7.515337	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
22	7.515357	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
23	7.995685	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
24	8.514389	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
25	8.514410	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
26	9.517402	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
27	9.517424	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
28	9.994917	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
29	10.517218	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
30	10.517241	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
31	11.517254	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
32	11.517274	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
33	11.994740	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
34	12.517208	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
35	12.517231	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
36	13.517204	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
37	13.517227	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
38	13.994680	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
39	14.517174	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
40	14.517197	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
41	15.517300	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
42	15.517322	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
43	15.995055	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
44	16.517248	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
45	16.517268	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
46	17.517171	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
47	17.517194	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
48	17.993427	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
49	18.517142	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
50	18.517165	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply

## Diferenciamos los distintos tipos de paquetes que se ven aquí:

- Están los BPDUs utilizados por el protocolo Spanning Tree, si nos fijamos la columna source podemos saber la MAC del origen que en este caso es switch2.
- Los paquetes ICMP son producidos por el ping desde PC2 a PC3.
- También observamos los paquetes ARP request y ARP reply. El primero pregunta por broadcast a que dirección MAC pertenece una IP y el segundo le responde, pudiendo ser la propia máquina u otra que la conozca.

Por ejemplo en la fila 2,3: PC2 pregunta por 172.31.0.3 y es PC3 quien contesta.

En la fila 17,18: PC3 pregunta por 172.31.0.2, y es PC2 quien contesta.

Cuando se mandan los ARP request (pc2) se actualizan las tablas de los switch1 y switch2 con la MAC de pc2.

Luego se realiza un ARP reply (pc3) por lo que switch2 actualiza su tabla pues conoce la MAC de pc3.

Luego se realiza el ping entre pc2 y pc3, y pc3 responde a pc2.

Switch1 no conoce a pc3 pues el tráfico fue filtrado por el switch2.

Observemos las tablas ahora:

```
switch1:~# brctl showmacs br0
port no mac addr      is local?      ageing timer
  1      00:1f:40:00:01:00  yes             0.00
  2      00:1f:40:00:01:01  yes             0.00
  2      00:1f:40:00:20:00  no              24.97
switch1:~#
```

```
switch2:~# brctl showmacs br0
port no mac addr      is local?      ageing timer
  2      00:1f:40:00:01:01  no              0.30
  1      00:1f:40:00:02:00  yes             0.00
  2      00:1f:40:00:02:01  yes             0.00
  1      00:1f:40:00:20:00  no              0.41
  1      00:1f:40:00:30:00  no              0.41
switch2:~#
```

Switch1 ahora conoce a pc2.

Switch2 ahora conoce a pc2 y pc3.

pc1 no es conocido por ningún switch ya que el único mensaje que le llega es un ARP request de PC2 en broadcast. Luego de que Switch2 conoce a pc2 y a pc3, este filtra el contenido de estas pc al resto de la red.

### 2. ¿Cuál es el tiempo de vida (lifetime) por defecto de las entradas en las tablas de direcciones MAC de ambos switches? Configure su valor en 3 segundos.

Utilizo el comando "brctl showstp br0" en ambos switches para obtener el lifetime por defecto. Este se muestra en el campo "ageing time" y es de 300 segundos.

Para cambiar este valor ejecuto en ambos switches el comando: "brctl setageing br0 3" para setearlo en 3 segundos.

**3. Luego que las entradas correspondientes a las direcciones MAC de los pcs hayan desaparecido de las tablas de los switches, vuelva a generar tráfico desde pc2 hacia pc3 (utilizando nuevamente el comando ping) pero ahora forzando que cada mensaje ICMP echo (Request) se envíe cada 7 segundos; previamente y utilizando nuevamente el comando tcpdump, capture el tráfico en pc1 y en pc3 en los archivos /hosthome/ping.pc1.g3.cap y /hosthome/ping.pc3.g3.cap respectivamente. Luego de unos 30 segundos de estar generando tráfico, detenga el ping y las capturas. Analice con wireshark las capturas realizadas y explique el comportamiento observado. Almacene los archivos del laboratorio modificado en el directorio IRC-lab\_two-switches.**

#### Capturando tráfico en pc1 y pc3

**Para pc1:** tcpdump -i eth0 -w /hosthome/ping.pc1.g3.cap

**Para pc3:** tcpdump -i eth0 -w /hosthome/ping.pc3.g3.cap



Genero tráfico desde pc2 hacia pc3 cada 7 segundos

ping -i 7 173.31.0.3

Captura para pc1

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
2	0.302129	Speaker_00:20:00	Broadcast	ARP	Who has 172.31.0.3? Tell 172.31.0.2
3	2.008931	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
4	4.008975	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
5	5.300171	Speaker_00:30:00	Speaker_00:20:00	ARP	Who has 172.31.0.2? Tell 172.31.0.3
6	6.009480	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
7	8.009604	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
8	10.007415	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
9	12.007221	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
10	14.007065	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
11	14.312509	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
12	16.007126	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
13	18.007264	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
14	20.009412	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
15	21.311192	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
16	22.009320	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
17	24.009478	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
18	26.009366	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
19	28.009230	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
20	28.313336	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
21	30.009234	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
22	32.009207	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
23	33.310878	Speaker_00:30:00	Speaker_00:20:00	ARP	Who has 172.31.0.2? Tell 172.31.0.3
24	34.009374	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
25	36.009396	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
26	38.009140	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
27	40.009166	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001
28	42.009209	Speaker_00:01:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 0 Port = 0x8001

Se ve que cada 7 segundos se envían paquetes desde pc2 del comando ping hacia pc3 (se ven los ICMP-request pero no los ICMP-reply).

Al cambiar el ageing time a 3 segundos la tabla se limpia en ese tiempo, mientras el ping se realiza cada 7 segundos. Por lo que entre ping y ping se limpia la tabla.

Estamos capturando desde pc1 por lo que podemos ver los echo requests de pc2 pues al hacerlo se lo está pidiendo al switch2, y como ya limpio su tabla no sabe dónde está pc3 por lo que hace un broadcast a todas las maquinas.

El response no lo ve pc1 pues cuando pc2 envió el mensaje quedo registrado en switch2, y luego pc3 envía el echo response a pc2 y todavía el switch no limpio la tabla por lo que solamente le llega a pc2. Switch2 filtra el tráfico hacia el switch1 y pc1 (filtra los echo reply y los ARP responses)

También vemos paquetes ARP y del Spanning Tree que ya comentamos anteriormente.

## Captura para pc3

No. ↓	Time	Source	Destination	Protocol	Info
1	0.000000	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
2	1.999993	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
3	2.302029	Speaker_00:20:00	Broadcast	ARP	Who has 172.31.0.3? Tell 172.31.0.2
4	2.302101	Speaker_00:30:00	Speaker_00:20:00	ARP	172.31.0.3 is at 00:1f:40:00:30:00
5	2.302832	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
6	2.302992	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
7	4.008990	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
8	6.008986	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
9	7.299897	Speaker_00:30:00	Speaker_00:20:00	ARP	Who has 172.31.0.2? Tell 172.31.0.3
10	7.300215	Speaker_00:20:00	Speaker_00:30:00	ARP	172.31.0.2 is at 00:1f:40:00:20:00
11	8.009502	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
12	9.313188	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
13	9.313217	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
14	10.009645	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
15	12.007541	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
16	14.007247	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
17	16.007096	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
18	16.312300	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
19	16.312322	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
20	18.007156	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
21	20.007300	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
22	22.009434	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
23	23.311239	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
24	23.311260	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
25	24.009401	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
26	26.009356	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
27	28.009388	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
28	30.009292	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
29	30.313226	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
30	30.313250	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
31	32.009273	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
32	34.009248	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
33	35.310466	Speaker_00:30:00	Speaker_00:20:00	ARP	Who has 172.31.0.2? Tell 172.31.0.3
34	35.310837	Speaker_00:20:00	Speaker_00:30:00	ARP	172.31.0.2 is at 00:1f:40:00:20:00
35	36.009334	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
36	37.313102	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
37	37.313158	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
38	38.009247	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
39	40.009218	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
40	42.009227	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
41	44.009333	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
42	44.312196	172.31.0.2	172.31.0.3	ICMP	Echo (ping) request
43	44.312222	172.31.0.3	172.31.0.2	ICMP	Echo (ping) reply
44	46.001816	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001
45	48.009306	Speaker_00:02:00	Spanning-tree-(for-bridges)_00	STP	Conf. Root = 32768/0/00:1f:40:00:01:00 Cost = 100 Port = 0x8001

En este caso sí podemos ver tanto los ARP responses como los echo replies entre pc2 y pc3 ya que están en el mismo dominio de colisión (C).

De esta forma comprobamos como los switches filtran los paquetes que no son necesarios para el otro segmento de la red. En este caso el switch2 está filtrando los paquetes entre pc2 y pc3 hacia el resto de la red (switch1 y pc1).