

# ***Obligatorio 3***

Redes de Computadores 2020

---

Grupo 14 - Eric Pintos, Petter Boussard, Nadia Recarey

Facultad de Ingeniería - UDELAR

# Índice

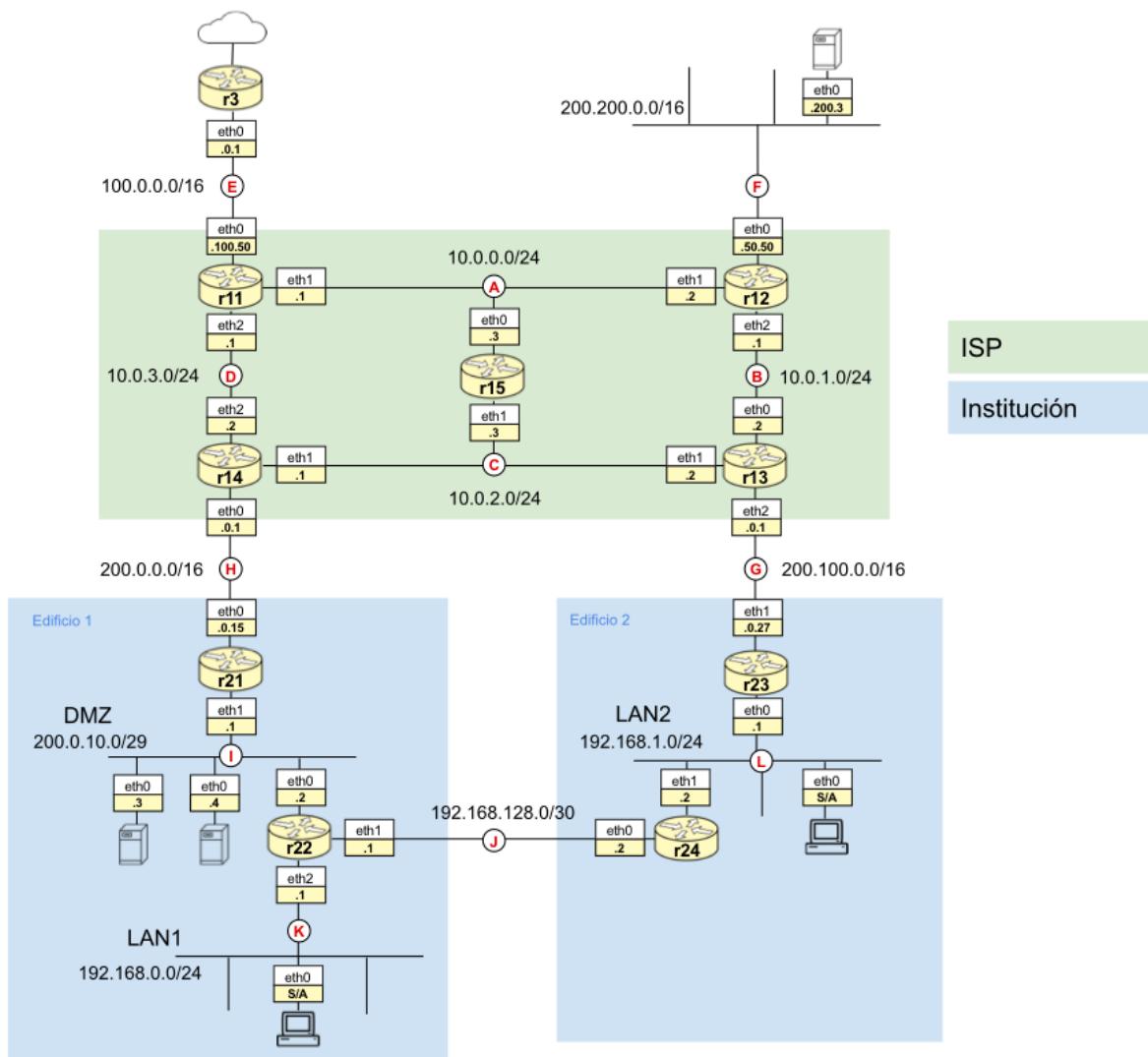
<b>Índice</b>	<b>2</b>
<b>Parte 0: Numerar la topología</b>	<b>4</b>
Topología de Red	4
Configuración de la Red	4
<b>Parte 1: Configurar ISP</b>	<b>5</b>
Entregable 1.1	5
Configuración estática	5
Verificación experimental de la configuración	5
Entregable 1.2	8
Configuración OSPF	8
Verificación experimental del inicio de OSPFD	9
Deshabilitar una interfaz	14
<b>Parte 2: Configurar Institución</b>	<b>17</b>
Análisis de NAT y DHCP	17
NAT	17
DHCP	17
Configuración Propuesta	18
Implementación en la Red	18
Configuración	18
Rutas estáticas en los routers de los edificios	18
Configuraciones adicionales	19
Configuración de NAT	20
Configuración de DHCP	20
Verificación	22
Verificación de tráfico general	22
NAT	24
DHCP	27
Análisis	27
NAT	27

DHCP	28
<b>Parte 3: Institución con Switches</b>	<b>30</b>
Implementación en la Red	30
Configuración	30
Tráfico de LAN3 a Internet	31
Tráfico de LAN3 a DMZ	32
Tráfico de Internet a la DMZ	33
Estudio Comparativo	33
Comparación Partes 2 y 3	33
Self-learning	37
Funciones de un switch (Filtrado y Reenvío)	37
Prueba de Self-learning	39

## **Parte 0: Numerar la topología**

El objetivo de esta parte es asignar direcciones IP a todos los dispositivos de la red. Para esto, como primer paso analice el contenido del laboratorio `redes-ob3`, en particular la definición de la topología dada en el archivo `lab.conf`.

*Topología de Red*



## *Configuración de la Red*

En el entregable, en la carpeta Entregable0 se encuentran los archivos (.startup) con las direcciones configuradas en las interfaces correspondientes.

Las configuraciones de realizan utilizando ifconfig, por ejemplo para asignarle la IP correspondiente a la interfaz eth0 de r11 se agrega el siguiente comando al archivo r11.startup:

```
ifconfig eth0 100.0.100.50 netmask 255.255.0.0 broadcast 100.0.255.255 up
```

# Parte 1: Configurar ISP

El ISP debe satisfacer los siguientes requerimientos:

- El ISP está compuesto por un backbone de 5 routers (r1\*).
- Debe poder enrutar entre todas las subredes públicas conectadas al backbone (p. ej. entre 200.0.0.0/16 y 200.200.0.0/16).
- El resto de Internet está disponible a través del router r3.

## Entregable 1.1

### Configuración estática

En la carpeta Entregable1.1 se encuentran los archivos .startup de cada router con las rutas estáticas configuradas.

Por ejemplo, en el archivo startup del router r11, configuramos la ruta de los paquetes con destino 200.200.0.0/16 para que reenvíe los paquetes al router r12. Esto lo hicimos agregando el siguiente comando en el archivo r11.startup:

```
route add -net 200.200.0.0 netmask 255.255.0.0 gw 10.0.0.2 dev eth11
```

Lo que quiere decir es que la subred 200.200.0.0/16 es accedida mediante 10.0.0.2 en la interfaz eth1.

### Verificación experimental de la configuración

En la carpeta Entregable1.1/capturas se encuentran las capturas de tráfico que realizamos para hacer las pruebas.

A fin de verificar experimentalmente la configuración, agregamos rutas por defecto en los routers r3, r21 y r23 y en el server12 para que estos se comuniquen directamente con el router del ISP con el cual están conectados. Esto nos facilitó la tarea de verificar la configuración, ya que ahora estos dispositivos saben por defecto que para cualquier intercambio de paquetes que deseen hacer, deberán consultar a su router del ISP.

Escuchamos el tráfico usando tcpdump en los routers r3, r21 y r23 y en el server12. En los archivos startup de cada uno de ellos incluimos el siguiente comando:

```
tcpdump -i eth* -s 65335 -w shared/r**.pcap
```

---

<sup>1</sup> [Kathara Lab Static Routing](#)

Donde <eth\*> es la interfaz donde estamos escuchando el tráfico, -s <size> indica el tamaño de los paquetes a escuchar y -w <directory> indica el directorio en donde vamos a guardar el archivo pcap que contiene la captura de tráfico.

Luego de levantar el laboratorio kathara con la configuración mencionada, realizamos traceroute desde cada uno de los hosts por fuera del backbone (server12, r3, r21 y r23) en los siguientes dispositivos:

- desde el server12 hacia r3, r23 y r21
- desde r3 hacia r23, r21 y al server12
- desde r23 hacia r3, r21 y al server12
- desde r21 hacia r3, r23 y al server12

Analizaremos a continuación solamente el primer caso (desde el server12 hacia los routers), ya que para los demás el análisis es análogo.

En la siguiente imagen vemos los resultados de realizar los tres traceroutes:

```
eric — root@server12: / — kathara ▾ kathara connect -l server12 — 80x24
++ ifconfig eth0 200.200.200.3 netmask 255.255.0.0 broadcast 200.200.255.255 up
++ route add default gw 200.200.50.50
++ tcpdump -i eth0 -s 65335 -w shared/server12.pcap
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65335 bytes
s

--- End Startup Commands Log

root@server12:/# traceroute 200.0.0.15 && traceroute 200.100.0.27 && traceroute
100.0.0.1
traceroute to 200.0.0.15 (200.0.0.15), 64 hops max
 1  200.200.50.50  0.007ms  0.003ms  0.003ms
 2  10.0.0.1  0.003ms  0.002ms  0.002ms
 3  10.0.2.1  0.004ms  0.003ms  0.002ms
 4  200.0.0.15  0.008ms  0.004ms  0.005ms
traceroute to 200.100.0.27 (200.100.0.27), 64 hops max
 1  200.200.50.50  0.105ms  0.004ms  0.003ms
 2  10.0.1.2  0.003ms  0.003ms  0.003ms
 3  200.100.0.27  0.003ms  0.003ms  0.003ms
traceroute to 100.0.0.1 (100.0.0.1), 64 hops max
 1  * 200.200.50.50  0.007ms  0.008ms
 2  10.0.0.1  0.010ms  0.005ms  0.007ms
 3  100.0.0.1  0.005ms  0.045ms  0.005ms
root@server12:/#
```

Se puede observar que para enviar un paquete desde el server12 hacia el router r21 (200.0.0.15), pasa primero por r12 (200.200.50.50), luego por r11 (10.0.0.1), luego por r14 (10.0.2.1) y finalmente a r21 (200.0.0.15). Esto coincide con el comportamiento esperado.

Para enviar paquetes desde el server12 al router r23, el camino que hace el paquete es r12, r13 y finalmente r23. Y finalmente para enviar paquetes desde el server12 hacia el router r3, el

paquete pasa primero por r12, luego por r11 y luego por r3. Estas rutas también coinciden con el comportamiento esperado.

Para finalizar esta sección realizaremos una breve demostración de las capturas solicitadas, las cuales se encuentran en la carpeta capturas de este entregable.

No.	Time	Source	Destination	Protocol	Length	Info
16	198.771316	200.200.200.3	100.0.0.1	UDP	51	39953 - 33436 Len=9
17	198.771335	100.0.0.1	200.200.200.3	ICMP	79	Destination unreachable (Port unreachable)
18	198.771386	200.200.200.3	100.0.0.1	UDP	51	39953 - 33436 Len=9
19	198.771393	100.0.0.1	200.200.200.3	ICMP	79	Destination unreachable (Port unreachable)
20	198.771436	200.200.200.3	100.0.0.1	UDP	51	39953 - 33436 Len=9
21	198.771442	100.0.0.1	200.200.200.3	ICMP	79	Destination unreachable (Port unreachable)
45	231.055348	200.200.200.3	100.0.0.1	ICMP	79	Destination unreachable (Port unreachable)
47	231.055462	200.200.200.3	100.0.0.1	ICMP	79	Destination unreachable (Port unreachable)
49	231.055618	200.200.200.3	100.0.0.1	ICMP	79	Destination unreachable (Port unreachable)

No.	Time	Source	Destination	Protocol	Length	Info
16	188.976810	200.200.200.3	200.100.0.27	UDP	51	42866 - 33436 Len=9
17	188.977196	200.100.0.27	200.200.200.3	ICMP	79	Destination unreachable (Port unreachable)
18	188.977607	200.200.200.3	200.100.0.27	UDP	51	42866 - 33436 Len=9
19	188.977629	200.100.0.27	200.200.200.3	ICMP	79	Destination unreachable (Port unreachable)
20	188.977944	200.200.200.3	200.100.0.27	UDP	51	42866 - 33436 Len=9
21	188.977954	200.100.0.27	200.200.200.3	ICMP	79	Destination unreachable (Port unreachable)
35	207.790170	200.200.200.3	200.100.0.27	ICMP	79	Destination unreachable (Port unreachable)
37	207.790222	200.200.200.3	200.100.0.27	ICMP	79	Destination unreachable (Port unreachable)
39	207.790267	200.200.200.3	200.100.0.27	ICMP	79	Destination unreachable (Port unreachable)

Frame 16: 51 bytes on wire (408 bits), 51 bytes captured (408 bits)  
 ► Ethernet II, Src: 02:42:ac:19:00:02 (02:42:ac:19:00:02), Dst: 02:42:ac:19:00:03 (02:42:ac:19:00:03)  
 ► Internet Protocol Version 4, Src: 200.200.200.3, Dst: 200.100.0.27  
 0000 02 42 ac 19 00 03 02 42 ac 19 00 02 08 00 45 00 ·B...B.....E·  
 0010 00 25 fa b8 40 00 01 11 25 c4 c8 c8 03 c8 64 ·%@...%...·  
 0020 00 1b a7 72 82 9c 00 11 59 6e 53 55 50 45 52 4d ·r...YnSUPERM·

En ambos casos estamos filtrando por la IP del server12 (200.200.200.3). Podemos ver como el tráfico de Traceroute (tanto las solicitudes UDP como ICMP) llegan exitosamente a ambos

routers desde el server12. Esto significa que la comunicación dentro del ISP está funcionando tal y como esperamos.

## **Entregable 1.2**

### **Configuración OSPF**

A modo de repaso teórico, OSPF es un protocolo de routing para redes IP.<sup>2</sup> En este, cada router propaga su estado local (interfaces, vecinos alcanzables) a través de la red utilizando List State Advertisements. Basado en esta información, cada router construye y mantiene una base de datos de Link States que describe toda la topología de la red, la cual usa para computar su camino más corto hacia otro router. Es posible asignar costos a cada interfaz, aunque para esta sección decidimos que todos tengan el mismo costo (por defecto es 10).

Para implementar OSPF en el laboratorio se utilizó Zebra/Quagga, un software que permite utilizar varios protocolos de ruteo, entre ellos OSPF. Este nos provee un servidor al cual nos podemos conectar por telnet para inspeccionar las rutas e interfaces de cada router.<sup>3</sup>

En la carpeta Entregable1.2 se encuentran los archivos con la configuración de OSPF solicitada. Esta se realizó en base al laboratorio proporcionado en la letra<sup>4</sup>. A continuación explicaremos cómo fue realizada esta configuración.

Todos los routers del ISP tienen en su directorio /etc/quagga dos archivos:

- **daemons:** indica los protocolos de ruteo que debe soportar el router en cuestión
- **ospfd.conf:** configura el demonio de ospfd que habilitamos en daemons

El contenido de **daemons** es el siguiente:

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

En nuestro caso habilitamos zebra y ospfd. zebra indica que el daemon principal de Zebra/Quagga será inicializado, lo cual necesitamos si queremos inspeccionar lo que sucede en las tablas del router, y ospfd habilita el daemon de OSPF. No es necesario habilitar ospf6d ya que esta está orientada a direcciones IPv6 que no estamos utilizando en este laboratorio.

---

<sup>2</sup> [Kathara Labs: OSPF](#)

<sup>3</sup> [Kathara Labs: Quagga](#)

<sup>4</sup> [Kathara Wiki: Interdomain Routing](#)

El contenido de **ospfd.conf** es el siguiente:

```
hostname ospfd
password zebra
enable password zebra
!
router ospf
!
! Speak OSPF on all interfaces falling in 10.0.0.0/16
network 10.0.0.0/16 area 0.0.0.0
redistribute connected
!
log file /var/log/zebra/ospfd.log
!
```

El propósito de este archivo es configurar la autenticación del demonio zebra, habilitar ospf, configurar las redes conectadas al router y habilitar los logs.

Redistribute connected es el encargado de decirle a OSPF que tiene que comunicarle a sus routers conectados, la información que conoce de la red.<sup>5</sup>

Decidimos omitir la asignación de costo para cada interfaz, ya que no se requería ponderar ningún enlace por sobre otro. En caso de querer hacerlo, es posible con las siguientes líneas:

```
interface <interfaz (p. ej: eth0)>
ospf cost <costo (p. ej: 20)>
```

Finalmente, pero no menos importante, cada uno de los **.startup** de los routers del ISP invoca zebra, el cual se encarga de levantar esta configuración:

```
/etc/init.d/zebra start
```

### Verificación experimental del inicio de OSPFD

En la carpeta Entregable1.2/capturas/b se encuentran las capturas solicitadas.

Dentro de los archivos startup de cada router del backbone, iniciamos la captura de tráfico mediante tcpdump, como hablamos anteriormente. Luego de esperar 30 segundos, cerramos el laboratorio kathara (mediante el comando *kathara lclean*) para finalizar las capturas. En estas capturas se pueden observar varios paquetes del protocolo OSPF.

---

<sup>5</sup> [Zebra and OSPF](#)

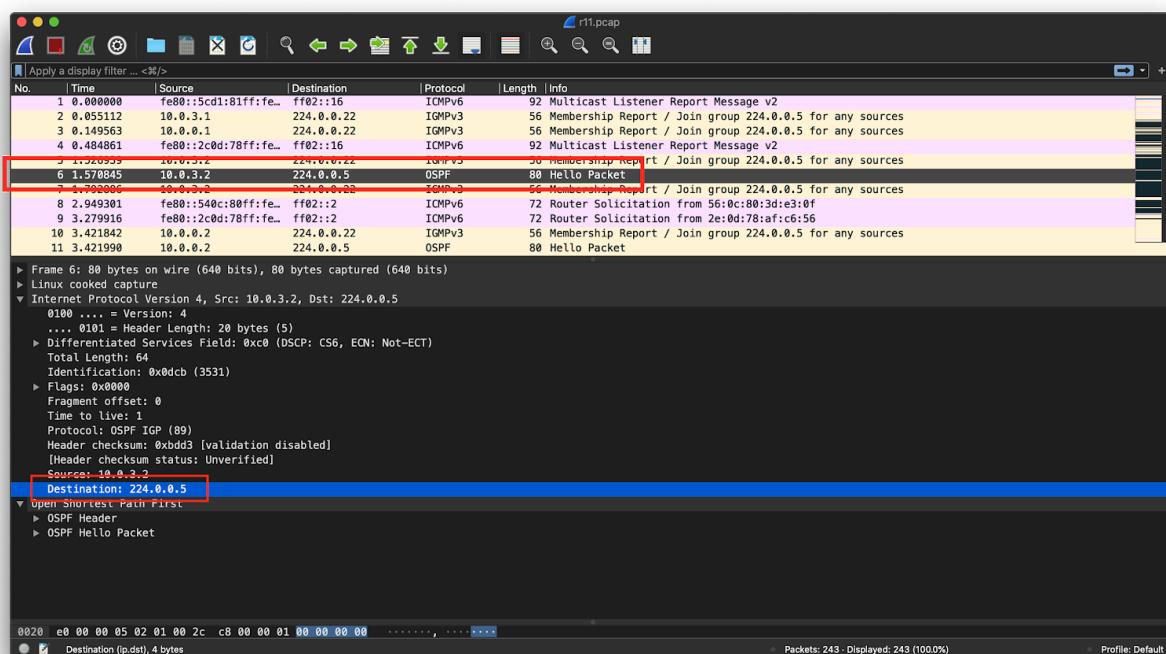
Hay 5 tipos de paquetes **Link State Advertisement** (de ahora en adelante LSA) en la versión 2 de OSPF:

1. Hello packet
2. Database Descriptor packet
3. Link State Request packet
4. Link State Update packet
5. Link State Acknowledgment packet

Los paquetes **Hello** se envían periódicamente a todas las interfaces (incluidos los enlaces virtuales) para descubrir, establecer y mantener relaciones con routers vecinos.

Una importante característica de OSPF es el uso de IPs Multicast. Los paquetes Hello son enviados siempre con **destino 224.0.0.5** (AllSPFRouters), una de las IP multicast de OSPF. Todos los routers que corran OSPF deben estar preparados para recibir paquetes con este destino. Estos paquetes no deben reenviarse, están diseñados para ser “single hop only” por lo que su TTL siempre debe ser 1.<sup>6</sup>

En la siguiente imagen puede verse la IP destino de uno de los Hello packets que observamos en las capturas que realizamos.



Es importante que todos los routers mantengan actualizadas las bases de datos de enlaces y los paquetes **Database Descriptor (DBD)** se utilizan para sincronizarlas.

<sup>6</sup> [RFC 2328](#)

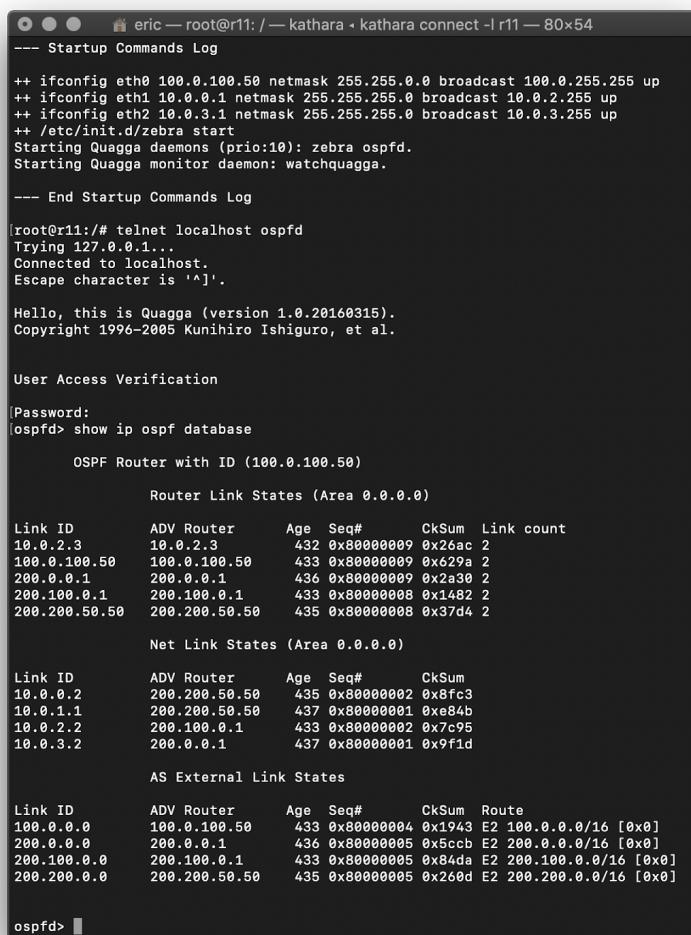
Luego del intercambio de paquetes DBD, un router puede encontrar que no tiene actualizada su base de datos, por lo que utiliza paquetes **Link State Request (LS Request)** para solicitar a los routers vecinos, bases de datos que estén más actualizadas.

Los paquetes **Link State Update (LS Update)** implementan la saturación o flooding de paquetes LSA. Cada paquete LSA contiene rutas, métricas e información de la topología de una parte de la red OSPF. El router local envía LSAs dentro de paquetes LS Update a todos los routers vecinos. Además envía LS Updates en respuesta a LS Requests.

Los paquetes **Link State Acknowledgment (LS Acknowledgement)** se utilizan como acuse de recibo de cada LSA. Un solo LSAck puede servir como recibo de varios LSAs.<sup>7</sup>

### Telnet localhost ospfd

Para ejecutar el comando `show ip ospf database` ofrecido por la interfaz de `ospfd`, nos conectamos por telnet para acceder primero a la interfaz. En la carpeta Entregable1.2/capturas/c se encuentran screenshots con los resultados de la ejecución del comando en cada uno de los routers del backbone. En la siguiente imagen vemos el resultado para el router r11.



```
eric — root@r11: / — kathara -> kathara connect -l r11 — 80x54
--- Startup Commands Log
++ ifconfig eth0 100.0.100.50 netmask 255.255.0.0 broadcast 100.0.255.255 up
++ ifconfig eth1 10.0.0.1 netmask 255.255.255.0 broadcast 10.0.2.255 up
++ ifconfig eth2 10.0.0.1 netmask 255.255.255.0 broadcast 10.0.3.255 up
++ /etc/init.d/zebra start
Starting Quagga daemons (priorities: zebra ospfd).
Starting Quagga monitor daemon: watchquagga.

--- End Startup Commands Log
[root@r11:/# telnet localhost ospfd
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is Quagga (version 1.0.20160315).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

[Password:
[ospfd> show ip ospf database

        OSPF Router with ID (100.0.100.50)

                Router Link States (Area 0.0.0.0)
                Link ID      ADV Router      Age Seq#      CkSum  Link count
                10.0.2.3    10.0.2.3      432 0x80000009 0x26ac 2
                100.0.100.50 100.0.100.50  433 0x80000009 0x629a 2
                200.0.0.1    200.0.0.1      436 0x80000009 0x2a30 2
                200.100.0.1  200.100.0.1   433 0x80000008 0x1482 2
                200.200.50.50 200.200.50.50 435 0x80000008 0x37d4 2

                Net Link States (Area 0.0.0.0)
                Link ID      ADV Router      Age Seq#      CkSum
                10.0.0.2    200.200.50.50  435 0x80000002 0xbfc3
                10.0.1.1    200.200.50.50  437 0x80000001 0xe84b
                10.0.2.2    200.100.0.1    433 0x80000002 0x7c95
                10.0.3.2    200.0.0.1     437 0x80000001 0x9f1d

                AS External Link States
                Link ID      ADV Router      Age Seq#      CkSum  Route
                100.0.0.0    100.0.100.50  433 0x80000004 0x1943 E2 100.0.0.0/16 [0x0]
                200.0.0.0    200.0.0.1     436 0x80000005 0x5ccb E2 200.0.0.0/16 [0x0]
                200.100.0.0  200.100.0.1   433 0x80000005 0x84da E2 200.100.0.0/16 [0x0]
                200.200.0.0  200.200.50.50 435 0x80000005 0x260d E2 200.200.0.0/16 [0x0]

[ospfd> ]
```

<sup>7</sup> [OSPF packet types](#)

En la primera línea vemos lo siguiente: OSPF Router with ID (100.0.100.50). Este ID es el ID del router en cuestión.

Luego pueden observarse tres tablas. Cada una de ellas muestra la información obtenida sobre la topología de la red mediante los LSAs.

La primer tabla Router Link States contiene la información obtenida utilizando los LSA del tipo 1, dentro del área (el área 0.0.0.0 es el backbone<sup>8</sup>). Los LSA de este tipo son generados por cada router hacia cada router del área a donde pertenece. Describen los estados de los enlaces entre los routers del área.

A continuación explicaremos el significado de cada columna:

- Link ID - Router ID (no es lo mismo que la dirección IP del router) o ID del nodo en el grafo correspondiente al área.
- ADV Router (Advertisement Router) - Router ID del dispositivo que publicó el LSA.
- Age - Tiempo en segundos desde que se recibió la información correspondiente a este LSA.
- Seq# - Número de secuencia del LSA, permite detectar información vieja o duplicada.
- Checksum - Valor de verificación del contenido completo del LSA.
- Link count - Cantidad de interfaces que participan del área, detectadas en el dispositivo.

La segunda tabla muestra los LSA de tipo 2, los Network Link States. Estos son generados por los DR (Designated Routers) y describen el conjunto de routers adjuntados a una red en particular. Esta tabla Indica cuáles son los dispositivos DR para el área y qué segmento de red representa cada uno.

En este caso la columna Link ID muestra la Dirección IP de la interfaz del router DR correspondiente al enlace que se reporta. Por ejemplo: 10.0.0.2 es la dirección IP de la interfaz del router 200.200.50.50 que conecta al segmento 10.0.0.0. El resto de las columnas indican lo mismo que en la tabla anterior.

La última tabla que vemos muestra AS External Link States LSAs, que son los de tipo 5. Son generados por routers ASBR (Autonomous System Boundary Router) y describen rutas a AS externos.

La última columna indica las subredes externas conectadas y el tipo de ruta externa mostrada (en este caso E2)<sup>9</sup>.

---

<sup>8</sup> [Show ospf database - juniper.net](https://www.juniper.net/techpubs/software/ospf/1.1/ospf-1.1.html)

<sup>9</sup> <https://www.ccexpert.us/ospf-2/e1-vs-e2-external-routes.html>

En la carpeta Entregable1.2/capturas/c/Zebra se encuentran los screenshots correspondientes a la conexión a zebra por telnet, y ejecutar el comando `sh ip route` en cada uno de los routers del backbone.

A continuación vemos el screenshot del router r11, para así compararla con la captura anterior.

```
eric — root@r11: / — kathara ▾ kathara connect -l r11 — 80x33
[eric@r11:~]# telnet localhost zebra
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.

Hello, this is Quagga (version 1.0.20160315).
Copyright 1996-2005 Kunihiro Ishiguro, et al.

User Access Verification

[Password:
[Router> sh ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
      O - OSPF, I - IS-IS, B - BGP, P - PIM, A - Babel,
      > - selected route, * - FIB route

O  10.0.0.0/24 [110/10] is directly connected, eth1, 00:15:21
C>* 10.0.0.0/24 is directly connected, eth1
O>* 10.0.1.0/24 [110/20] via 10.0.0.2, eth1, 00:15:19
O>* 10.0.2.0/24 [110/20] via 10.0.0.3, eth1, 00:15:12
      *           via 10.0.3.2, eth2, 00:15:12
O  10.0.3.0/24 [110/10] is directly connected, eth2, 00:15:25
C>* 10.0.3.0/24 is directly connected, eth2
C>* 100.0.0.0/16 is directly connected, eth0
C>* 127.0.0.0/8 is directly connected, lo
O>* 200.0.0.0/16 [110/20] via 10.0.3.2, eth2, 00:15:21
O>* 200.100.0.0/16 [110/20] via 10.0.0.2, eth1, 00:15:11
      *           via 10.0.0.3, eth1, 00:15:11
      *           via 10.0.3.2, eth2, 00:15:11
O>* 200.200.0.0/16 [110/20] via 10.0.0.2, eth1, 00:15:18
Router> ]
```

La principal diferencia entre ambos comandos es que en `ospfd` muestra la información recolectada via OSPF mostrando los enlaces descubiertos mediante los LSAs, y en `zebra` muestra las rutas y más información sobre las subredes, como el nombre de la red, máscara, por cuál router fue descubierto y por cuál interfaz está conectado.

Como puede observarse en las referencias de los códigos, O quiere decir que la subred fue descubierta mediante OSPF via qué router, en cuál interfaz y la hora (hace cuánto tiempo la conoce).

## Deshabilitar una interfaz

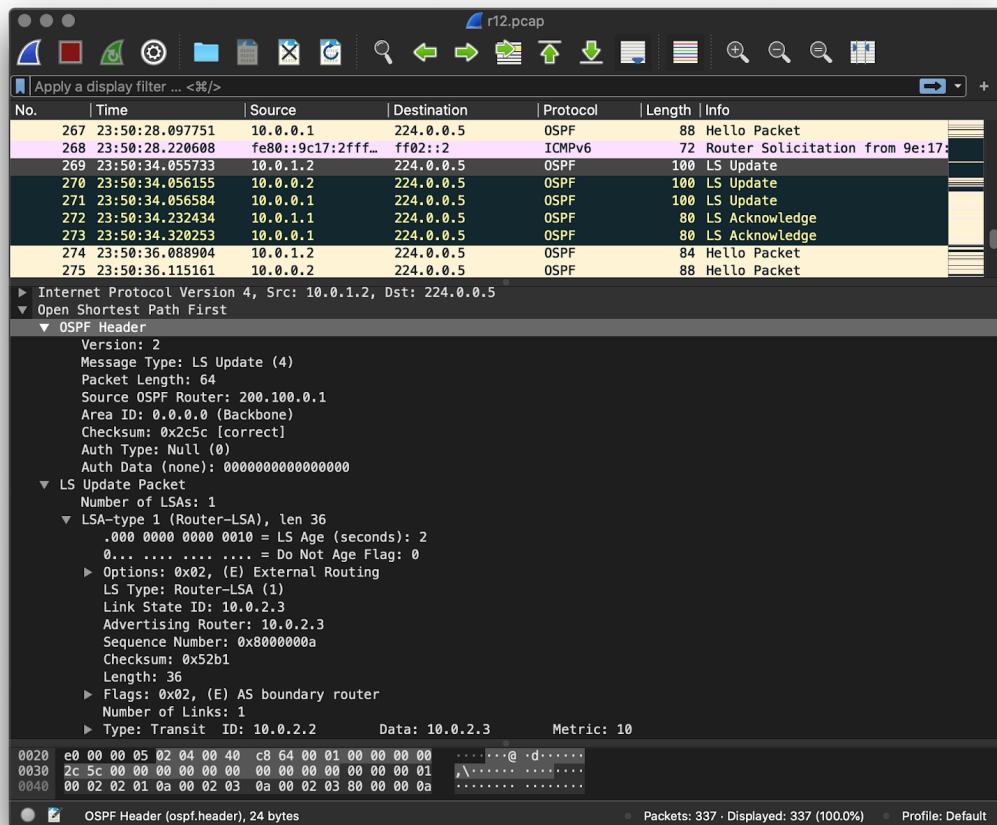
En la carpeta Entregable1.2/capturas/d se encuentran archivos que contienen las capturas de tráfico en cada uno de los routers del backbone, al tirar una interfaz de uno de ellos.

La prueba consistió en comenzar a escuchar el tráfico mediante tcpdump en cada uno de los routers del backbone. Luego de estabilizada la red, tiramos la interfaz eth0 del **router r15**. Esto lo realizamos ejecutando el siguiente comando en la consola del router:

```
ifconfig eth0 down
```

Cada vez que ocurre un cambio en un router, como lo es una baja de una interfaz se envían **LS Updates** a todos los routers del área con destino 224.0.0.5 (OSPF Multicast).

Si observamos la captura de tráfico del router **r12** vemos que primero llegan 3 paquetes **LS Update**.



En el header del primer paquete **LS Update** podemos ver la versión de OSPF (2), el tipo de mensaje (LS Update), el tamaño del paquete, el router ID del cual fue originado el LSA (200.100.0.1), el área (0.0.0.0), checksum y campos de autorización.

Luego en el cuerpo del mensaje vemos la cantidad de LSAs que han sido encapsulados dentro de este paquete, en este caso 1. En este mensaje vemos el tipo de LSA (tipo 1 Router LSA<sup>10</sup>) y algunas configuraciones y opciones. Vemos el Link State ID (10.0.2.3), el ADV router (10.0.2.3 - r15), el Seq#, checksum y tamaño. Este nuevo paquete indica en su payload que existe un enlace entre 10.0.2.2 (r13) y 10.0.2.3 (r15) con métrica 10. A modo de comparación, el contenido que tenía el LS Update que mandó anteriormente (nro. de paquete 163) tenía el siguiente payload:

```
Number of Links: 2
▼ Type: Transit ID: 10.0.0.2      Data: 10.0.0.3      Metric: 10
  Link ID: 10.0.0.2 - IP address of Designated Router
  Link Data: 10.0.0.3
  Link Type: 2 - Connection to a transit network
  Number of Metrics: 0 - TOS
  0 Metric: 10
▼ Type: Transit ID: 10.0.2.2      Data: 10.0.2.3      Metric: 10
  Link ID: 10.0.2.2 - IP address of Designated Router
  Link Data: 10.0.2.3
  Link Type: 2 - Connection to a transit network
  Number of Metrics: 0 - TOS
  0 Metric: 10
```

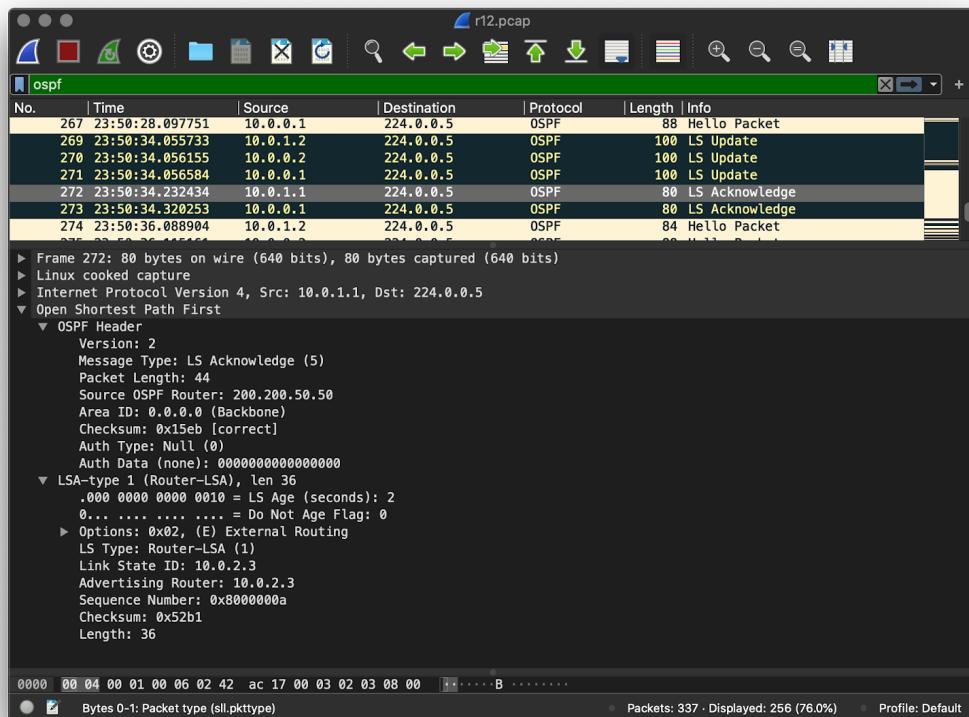
La principal diferencia aquí es que el LS Update de la 163 publicaba dos enlaces: uno entre 10.0.0.3 (r15) y 10.0.0.2 (r12), y otro entre 10.0.2.2 (r13) y 10.0.2.3 (r15). En el LS Update después de tirar la eth0 de r15 (10.0.0.3) vemos que esta no existe más: sólo se publica la de r15 a r13.

Lo mismo sucede en los dos paquetes LS Update siguientes: r15 publica sus enlaces nuevos tras la caída del enlace eth0.

A continuación, vemos como el r12 manda el LSAck en la captura siguiente. Como vemos contiene muchos campos iguales a los del LS Update anteriormente descrito. Particularmente, el que es de nuestro interés es el sequence number, el cual coincide en contenido.

---

<sup>10</sup> [OSPF LSA Types Explained](#)



Vemos varios intercambios de mensajes LS Update y LSAck debido a que cada router continúa enviando LS Updates si detecta que hay diferencias entre las demás bases de datos y la suya, hasta que sean todas iguales. También se puede observar que los mensajes Hello se siguen enviando periódicamente.<sup>11</sup><sup>12</sup>

<sup>11</sup> [Kathara Labs OSPF](#)

<sup>12</sup> [OSPF Neighbor States Explained \(OSPF States\)](#)

## Parte 2: Configurar Institución

La configuración de la institución debe satisfacer los siguientes requerimientos:

- La institución está instalada en dos edificios, conectados por un enlace propio (r22-r24), y cada uno con salida propia a Internet (mediante r21 y r23).
- Los hosts en LAN1 y LAN2 deben poder acceder a Internet (p.ej. 200.200.200.3)
- La subred DMZ está numerada con direcciones públicas, y ofrece servicios (p. ej. 200.200.200.3 debe poder conectarse a 200.0.10.3).
- Los hosts de LAN2 deben poder acceder a los servidores de la DMZ sin salir a Internet.

### *Análisis de NAT y DHCP*

#### NAT

NAT (Network Address Translation) es un método que permite a un router modificar en tiempo real la dirección de red de un paquete en tránsito. Este mecanismo es utilizado en los routers IP para intercambiar paquetes entre dos redes que asignan direcciones mutuamente incompatibles. De esta forma se maneja un "espacio privado" de direcciones que no se expone resto de la red<sup>13 14</sup>. A fin de realizar esto, cada router que lo habilite maneja una NAT Translation Table en la cual mapea una dirección/puerto dentro de su red hacia una dirección/puerto diferentes que expone al resto de la red (dirección IP pública).

El router NAT no parece un router a ojos del mundo exterior. En su lugar, el router NAT se comporta de cara al exterior como un único dispositivo con una dirección IP única.

#### DHCP

DHCP (Dynamic Host Configuration Protocol) es un protocolo cliente-servidor que permite a un host obtener su dirección IP automáticamente, sin la intervención de un administrador de red. Este protocolo es flexible en el sentido de que un host puede recibir la misma dirección IP cada vez que se conecta a una red, o puede recibir una dirección IP temporal. DHCP también permite agregar otro tipo de configuraciones tales como la máscara de subred, la dirección de los routers disponibles para determinado host (gateway) y la dirección del servidor DNS disponible<sup>15</sup>. También es posible configurar DCHP para que además de enviar las configuraciones mencionadas anteriormente envíe configuraciones para que los dispositivos dirijan el tráfico de determinada subred a otro router, no necesariamente el default gateway. Hacemos uso de esta funcionalidad para hacer que el Edificio 2 se comunique con la DMZ directamente a través del enlace entre r22 y r24.

---

<sup>13</sup> Kurose, Ross - Computer Networking A Top-Down Approach: 7th Edition

<sup>14</sup> [Network address translation](#)

<sup>15</sup> Kurose, Ross - Computer Networking A Top-Down Approach: 7th Edition

## Configuración Propuesta

La configuración propuesta incluye habilitar NAT en los routers r23 y r22. La razón es que estos son el punto de contacto de los ordenadores de las LAN con el resto de la red. De esta forma evitamos que hayan direcciones incompatibles con el resto de la red y no se exponen las IPs de los ordenadores de las LAN. Además como sabemos 192.168.xxx.xxx son direcciones muy comunes en las redes hogareñas (se trata de un rango de IPs privadas). No hay necesidad que los ordenadores expongan su IP fuera de su LAN, como sí sucede en el caso de la DMZ que contiene servidores y sus IPs deberán ser conocidas para establecer conexión.

En el caso de DHCP, sus servidores deberán ser instalados en r22 y r23, por razones similares a las de NAT: son los routers que tienen contacto con las LAN y tienen computadoras que no tienen IPs asignadas, por lo cual tenemos que asignarlas dinámicamente con DHCP. Es parte de la motivación de DHCP: asignar direcciones dentro de una LAN.

Omitimos r21 por razones similares: sólo tiene servidores y routers con direcciones estáticas. El router r24 es excluido porque la asignación de IPs en LAN2 ya está cubierta por r23. Vimos que en la LAN entre los routers r22 y r24 no es posible conectar más dispositivos por lo que no sería necesario configurar DHCP para esta subred. Si fuese necesario, el DHCP instalado en r22 podría estar configurado para servir a esa subred también.

## Implementación en la Red

### Configuración

Rutas estáticas en los routers de los edificios

La configuración de las rutas estáticas para los routers de los edificios se realizó utilizando el mismo comando utilizado para especificar las rutas estáticas para los routers del ISP en la parte 0. (utilizando route add).

- **r21:** Indicamos a **r14** como default gateway
  - **r22:** Indicamos a **r21** como default gateway y además agregamos una entrada a la tabla de ruteo para dirigir el tráfico hacia la LAN 2 mediante **r24**.<sup>16</sup>
- Esto lo logramos agregando los siguientes comandos en **r22.startup**:

```
# el resto del tráfico mandarlo a r21
route add default gw 200.0.10.1 dev eth0

# tráfico hacia la LAN2 (192.168.1.0/24) pasa por r24
route add -net 192.168.1.0 netmask 255.255.255.0 gw 192.168.128.2 dev eth1
```

---

<sup>16</sup> Una alternativa a esto sería habilitar NAT en r24 en lugar de usar las IPs de la LAN2. De esta forma, r22 no tendría que conocer las direcciones de la LAN2, tan sólo la IP estática de r24.

- **r23**: Indicamos a **r13** como default gateway
- **r24**: Indicamos a **r22** como default gateway

Para asegurarnos de que el tráfico dirigido desde la LAN2 hacia la DMZ no salga a internet, realizamos la configuración de una ruta estática utilizando DHCP para que los propios hosts envíen los paquetes dirigidos a la DMZ directamente hacia el router **r24**, esto nos permite que no dupliquemos el tráfico en el enlace, ya que si por ejemplo todo el tráfico de los hosts se enviaran a **r23**, este router debería reenviar el tráfico dirigido hacia la DMZ por el mismo enlace hacia el router **r24**.

En este caso consideramos que no era necesario establecer rutas directas entre las LANs. Sin embargo, es algo que se podría hacer fácilmente agregando una ruta estática en el DHCP para que los hosts de la **LAN2** manden los paquetes con destino a la **LAN1** al router **r24** (recordemos que **r22** tiene una ruta estática hacia **LAN2** por lo cual los paquetes "de vuelta" podrían llegar bien).

#### Configuraciones adicionales

A fin de verificar el funcionamiento de nuestra red hicimos las siguientes configuraciones adicionales:

- Agregamos un **default gateway** en **r11** para enviar los paquetes con destino a internet al router **r3**, y a su vez, que esta configuración se propague a todos los routers del backbone (para no tener que configurarlos estáticamente en cada uno). Para esto agregamos la siguiente línea en el archivo **ospfd.conf**:

```
default-information originate always
```

- Agregamos una ruta estática en **r14** indicando que todo el tráfico dirigido hacia la **DMZ** debería ser enviada a **r21** de la siguiente manera:

```
route add -net 200.0.10.0 netmask 255.255.255.248 gw 200.0.0.15 dev eth0
```

Lo que sucedía era que cuando llegaba un paquete con destino a la **DMZ** a **r14**, el mismo necesitaba determinar la dirección MAC de destino, por esta razón se mandaba un paquete **ARP** a la subred 200.0.0.0/16 (ya que las direcciones de la **DMZ** están contenidas en esta) pero este mensaje **ARP** nunca se respondía, ya que los dispositivos de la **DMZ** no están conectados directamente a la subred 200.0.0.0/16.

## Configuración de NAT

Para la configuración de NAT hicimos uso del comando que se recomendaba usar en el Anexo 2 de la letra.

```
iptables -t nat -A POSTROUTING -s <red interna> -o <interfaz salida> -j SNAT --to <ip pública>
```

- r22

```
iptables -t nat -A POSTROUTING -s 192.168.0.0/24 -o eth0 -j SNAT --to 200.0.10.2 (LAN 1)
iptables -t nat -A POSTROUTING -s 192.168.128.0/30 -o eth0 -j SNAT --to 200.0.10.2 (Dom. J)
iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth0 -j SNAT --to 200.0.10.2 (LAN2)
```

- r23

```
iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth1 -j SNAT --to 200.100.0.27
```

## Configuración de DHCP

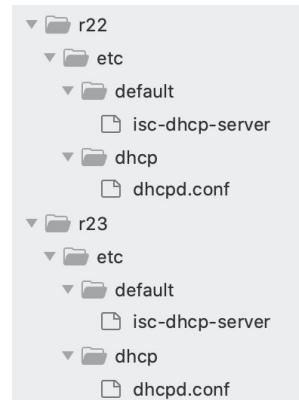
Para la configuración del servicio de DHCP en los routers **r22** y **r23** se utilizó una imagen del servidor DHCP que se obtuvo ejecutando el comando:

```
sudo docker pull mrichart/kathara:latest
```

Luego para indicarle a Kathara que utilice esa imagen, incluimos en el archivo **lab.conf** las siguientes líneas:

```
r22[image]="mrichart/kathara"
r23[image]="mrichart/kathara"
```

Los archivos de configuración de DHCP los podemos encontrar en las carpetas correspondientes a **r22** y **r23**, que tienen la siguiente estructura:



Dentro del archivo **isc-dhcp-server** especificamos la interfaz del router por la cual estará disponible el servicio de DHCP, por ejemplo para el router **r22** este archivo contiene:

```
INTERFACESv4="eth2"
```

**eth2** es la interfaz del router **r22** en la cual se encuentra la LAN1, dónde se conectarán las PCs y necesitarán que se les asigne una IP.

Por otro lado, en el archivo **dhcpd.conf** especificamos las distintas configuraciones del servidor DHCP, como por ejemplo el rango de dirección IP disponibles para asignar, rutas estáticas que serán configuradas en los hosts y también el default gateway.

Veamos el archivo **dhcpd.conf** para el router **r23**:

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.3 192.168.1.254;
}

# static route and default route
option rfc3442-classless-static-routes code 121 = array of integer 8;
option rfc3442-classless-static-routes 0, 192,168,1,1,
    29, 200,0,10,0, 192,168,1,2;
option ms-classless-static-routes code 249 = array of integer 8;
option ms-classless-static-routes 0, 192,168,1,1,
    29, 200,0,10,0, 192,168,1,2;
```

Lo que se realiza en este archivo es especificar el rango de direcciones que el servidor puede asignar a los hosts, además se encarga de configurar las rutas estáticas y por defecto.

El comando option \*-classless-static routes configura una ruta estática mediante DHCP<sup>17</sup>. Esto nos sirve para que, por ejemplo, para que el tráfico generado por la **pc24** dirigida a la **DMZ**, se envíe directamente al router **r24** ya que este tráfico no debe salir al ISP. Para rtear todo el tráfico que vaya hacia 200.0.10.0/29 (**DMZ**) a través del router 192.168.1.2 (**r24**) lo que debemos hacer es:

```
option ms-classless-static-routes 29, 200,0,10,0, 192,168,1,2;
option rfc3442-classless-static-routes 29, 200,0,10,0, 192,168,1,2;
```

---

<sup>17</sup> [Distributing static routes with DHCP](#)

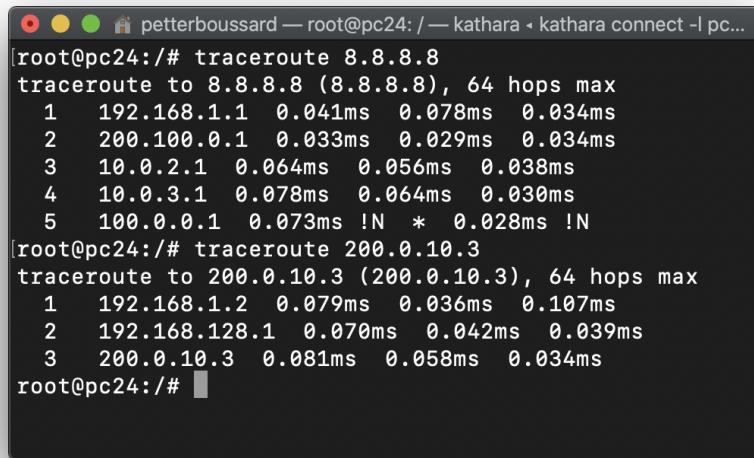
También se agregan rutas default. Esto es cuando tanto <MASCARA DESTINO> como <IP DESTINO> son 0. En este caso, todo el tráfico que no se dirija hacia la DMZ saldrá por 192.168.1.1 (**r23**) tal como indica la letra.

Cabe aclarar que el comando option \*-classless-static está duplicado por motivos de compatibilidad: uno es para Unix y el otro para Windows. Las líneas option \*-classless-static code son requeridas en esta configuración<sup>18</sup>.

## Verificación

Verificación de tráfico general

Para verificar que el tráfico desde la LAN1 y la LAN2 llega tanto a internet como a la DMZ, ejecutamos traceroute con destinos **8.8.8.8** y **200.0.10.3** respectivamente. Observemos primero los caminos que realizan los paquetes desde la **pc24** (que se encuentra en la LAN2)



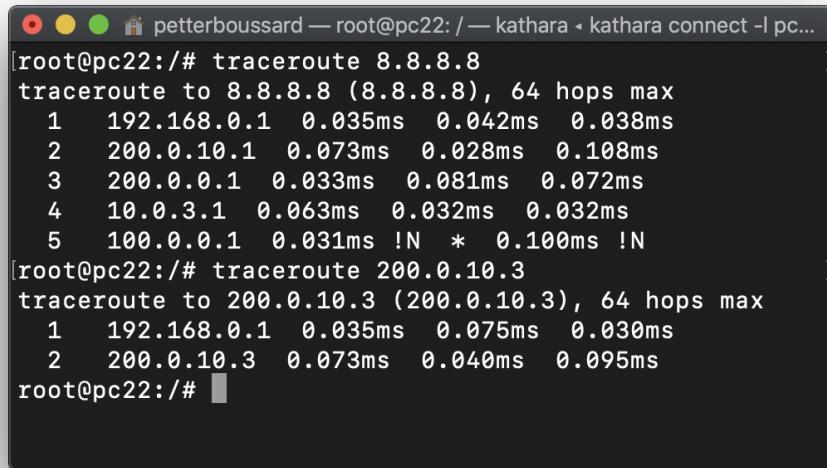
```
[root@pc24:/# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 64 hops max
 1  192.168.1.1  0.041ms  0.078ms  0.034ms
 2  200.100.0.1  0.033ms  0.029ms  0.034ms
 3  10.0.2.1  0.064ms  0.056ms  0.038ms
 4  10.0.3.1  0.078ms  0.064ms  0.030ms
 5  100.0.0.1  0.073ms !N *  0.028ms !N
[root@pc24:/# traceroute 200.0.10.3
traceroute to 200.0.10.3 (200.0.10.3), 64 hops max
 1  192.168.1.2  0.079ms  0.036ms  0.107ms
 2  192.168.128.1  0.070ms  0.042ms  0.039ms
 3  200.0.10.3  0.081ms  0.058ms  0.034ms
root@pc24:/# ]
```

Podemos observar que el traceroute llega al destino en ambos casos y se están tomando los caminos adecuados, el camino que hace un paquete desde la **pc24** hasta **8.8.8.8** es **r23, r13, r14, r11** y finalmente **r3**, luego **r3** debería comunicarse con internet, por ejemplo, usando el protocolo **BGP**. El camino que toma un paquete desde **pc24** hasta el **server21** (que se encuentra en la **DMZ**) es **r24, r22** y luego ya llega al destino. Con esto podemos verificar que se cumple la restricción de que los hosts de la LAN2 deben poder acceder a la **DMZ** sin tener que salir a internet.

Esto también verifica que la asignación de una IP a la **pc24** por parte del servidor DHCP se realizó correctamente, de otra forma no se hubiera podido realizar el traceroute exitosamente.

<sup>18</sup> [DHCP RFC3442 Classless Static Route Configuration - Abiquo Cloud Platform Documentation](#)

Analicemos ahora el traceroute desde la **pc22** hacia internet y hacia la DMZ:



```
[root@pc22:/# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 64 hops max
 1  192.168.0.1  0.035ms  0.042ms  0.038ms
 2  200.0.10.1  0.073ms  0.028ms  0.108ms
 3  200.0.0.1  0.033ms  0.081ms  0.072ms
 4  10.0.3.1  0.063ms  0.032ms  0.032ms
 5  100.0.0.1  0.031ms !N * 0.100ms !N
[root@pc22:/# traceroute 200.0.10.3
traceroute to 200.0.10.3 (200.0.10.3), 64 hops max
 1  192.168.0.1  0.035ms  0.075ms  0.030ms
 2  200.0.10.3  0.073ms  0.040ms  0.095ms
root@pc22:/# ]
```

Con esto podemos observar que todo funciona como se espera, el tráfico dirigido hacia internet llega hasta el router **r3** tomando el camino (**r22, r21, r14, r11, r3**) y el tráfico dirigido hacia la **DMZ** solo pasa por el router **r22** y llega a destino.

Con esto también logramos verificar que la asignación de **IP** a la **pc22** por parte del servidor DHCP se realizó correctamente, de lo contrario no se podría ejecutar el comando traceroute de forma exitosa.

Para verificar el tráfico de internet hacia la **DMZ** ejecutamos un **ping** y un **traceroute** desde el router **r3** hacia el servidor **server21** que se encuentra en la **DMZ**, logrando comprobar que todo se encuentra funcionando correctamente

```

petterboussard — root@r3: / — kathara ✧ kathara connect -l r3 — 71x21
[root@r3:/# ping 200.0.10.3
PING 200.0.10.3 (200.0.10.3) 56(84) bytes of data.
64 bytes from 200.0.10.3: icmp_seq=1 ttl=61 time=1.20 ms
64 bytes from 200.0.10.3: icmp_seq=2 ttl=61 time=0.261 ms
64 bytes from 200.0.10.3: icmp_seq=3 ttl=61 time=0.223 ms
64 bytes from 200.0.10.3: icmp_seq=4 ttl=61 time=0.211 ms
^C
--- 200.0.10.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3034ms
rtt min/avg/max/mdev = 0.211/0.476/1.209/0.423 ms
[root@r3:/#
[root@r3:/#
[root@r3:/# traceroute 200.0.10.3
traceroute to 200.0.10.3 (200.0.10.3), 30 hops max, 60 byte packets
 1  100.0.100.50 (100.0.100.50)  0.933 ms  0.131 ms  0.075 ms
 2  10.0.3.2 (10.0.3.2)  0.279 ms  0.085 ms  0.294 ms
 3  200.0.0.15 (200.0.0.15)  0.647 ms  0.317 ms  0.225 ms
 4  200.0.10.3 (200.0.10.3)  0.121 ms  0.122 ms  0.103 ms
root@r3:/#

```

## NAT

Para probar que NAT estuviera funcionando correctamente decidimos ejecutar el comando ping desde la **pc24** y la **pc22**, capturar el tráfico que llegaba a **r3** y a **server21** para lograr observar la dirección IP de origen del paquete ICMP echo request. Si logramos observar que la dirección IP de destino del paquete recibido era la dirección IP pública de los routers donde configuramos NAT (**r22** y **r23**), esto indicaba que NAT estaba funcionando correctamente.

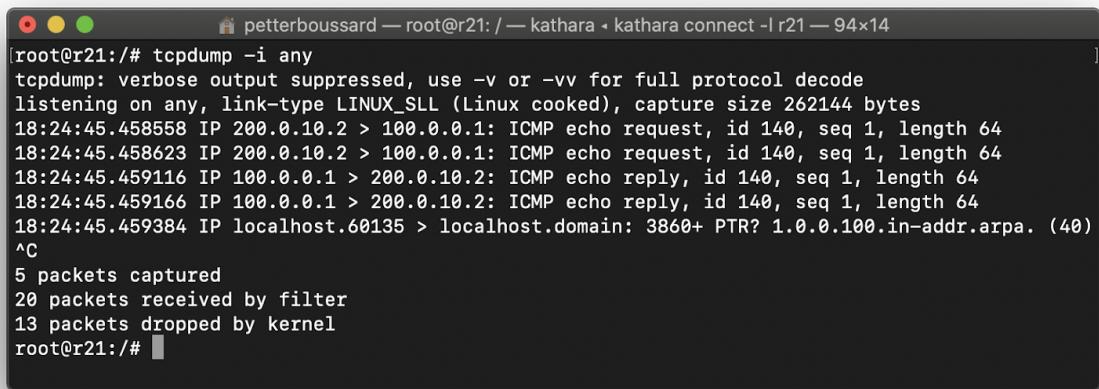
Veamos la ejecución del comando ping desde la **pc22** hacia **r3**:

```

petterboussard — root@pc22: / — kathara ✧ kathara connect -l pc...
[root@pc22:/# ping 100.0.0.1 -c 1
PING 100.0.0.1 (100.0.0.1) 56(84) bytes of data.
64 bytes from 100.0.0.1: icmp_seq=1 ttl=60 time=1.31 ms

--- 100.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.311/1.311/1.311/0.000 ms
root@pc22:/#

```



```
[root@r21:/# tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
18:24:45.458558 IP 200.0.10.2 > 100.0.0.1: ICMP echo request, id 140, seq 1, length 64
18:24:45.458623 IP 200.0.10.2 > 100.0.0.1: ICMP echo request, id 140, seq 1, length 64
18:24:45.459116 IP 100.0.0.1 > 200.0.10.2: ICMP echo reply, id 140, seq 1, length 64
18:24:45.459166 IP 100.0.0.1 > 200.0.10.2: ICMP echo reply, id 140, seq 1, length 64
18:24:45.459384 IP localhost.60135 > localhost.domain: 3860+ PTR? 1.0.0.100.in-addr.arpa. (40)
^C
5 packets captured
20 packets received by filter
13 packets dropped by kernel
root@r21:/#
```

Logramos ver que la dirección IP destino del paquete ICMP echo request que llegó a **r3** es **200.0.10.2** la cual es la dirección IP pública de **r22**, además, observamos que el paquete ICMP echo reply llegó correctamente a la **pc22**.

Luego se realizó una prueba similar desde la **pc24**, ejecutando ping a **server21** y a **server12**, y logramos verificar que el paquete ICMP que llegó a **server21** tenía una dirección IP de origen igual a **200.0.10.2 (r22)**, y la dirección de origen del paquete que llegó a **server12** era **200.100.0.27**, además como los paquetes ICMP echo reply fueron recibidos en la **pc24**, nos indica que que ambos routers donde configuramos NAT funcionan correctamente.

```

petterboussard — root@pc24: / — kathara - kathara connect -l pc24...
root@pc24:/# ping 200.0.10.3 -c 1
PING 200.0.10.3 (200.0.10.3) 56(84) bytes of data.
64 bytes from 200.0.10.3: icmp_seq=1 ttl=62 time=0.809 ms

--- 200.0.10.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.809/0.809/0.809/0.000 ms
root@pc24:/# ping 200.200.200.3 -c 1
PING 200.200.200.3 (200.200.200.3) 56(84) bytes of data.
64 bytes from 200.200.200.3: icmp_seq=1 ttl=61 time=1.16 ms

--- 200.200.200.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.161/1.161/1.161/0.000 ms
root@pc24:/#

```

```

petterboussard — root@server21: / — kathara - kathara connect -l server21 — 99x16
root@server21:/# tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
18:39:52.938479 IP 200.0.10.2 > 200.0.10.3: ICMP echo request, id 137, seq 1, length 64
18:39:52.938609 IP 200.0.10.3 > 200.0.10.2: ICMP echo reply, id 137, seq 1, length 64
18:39:52.939317 IP localhost.47172 > localhost.domain: 12549+ PTR? 3.10.0.200.in-addr.arpa. (41)
18:39:52.939360 IP localhost > localhost: ICMP localhost udp domain unreachable, length 77
^C
4 packets captured
18 packets received by filter
11 packets dropped by kernel
root@server21:/#

```

```

petterboussard — root@server12: / — kathara - kathara connect -l server12 — 100x15
root@server12:/# tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
18:40:49.106055 ARP, Request who-has 200.200.200.3 tell 200.200.50.50, length 28
18:40:49.106091 ARP, Reply 200.200.200.3 is-at e6:48:66:cd:65:a4 (oui Unknown). length 28
18:40:49.106170 IP 200.100.0.27 > 200.200.200.3: ICMP echo request, id 138, seq 1, length 64
18:40:49.106244 IP 200.200.200.3 > 200.100.0.27: ICMP echo reply, id 138, seq 1, length 64
18:40:49.107040 IP localhost.37469 > localhost.domain: 51890+ PTR? 3.200.200.200.in-addr.arpa. (44)
18:40:49.107668 IP localhost.59382 > localhost.domain: 52797+ PTR? 27.0.100.200.in-addr.arpa. (43)
^C
6 packets captured
28 packets received by filter
19 packets dropped by kernel
root@server12:/#

```

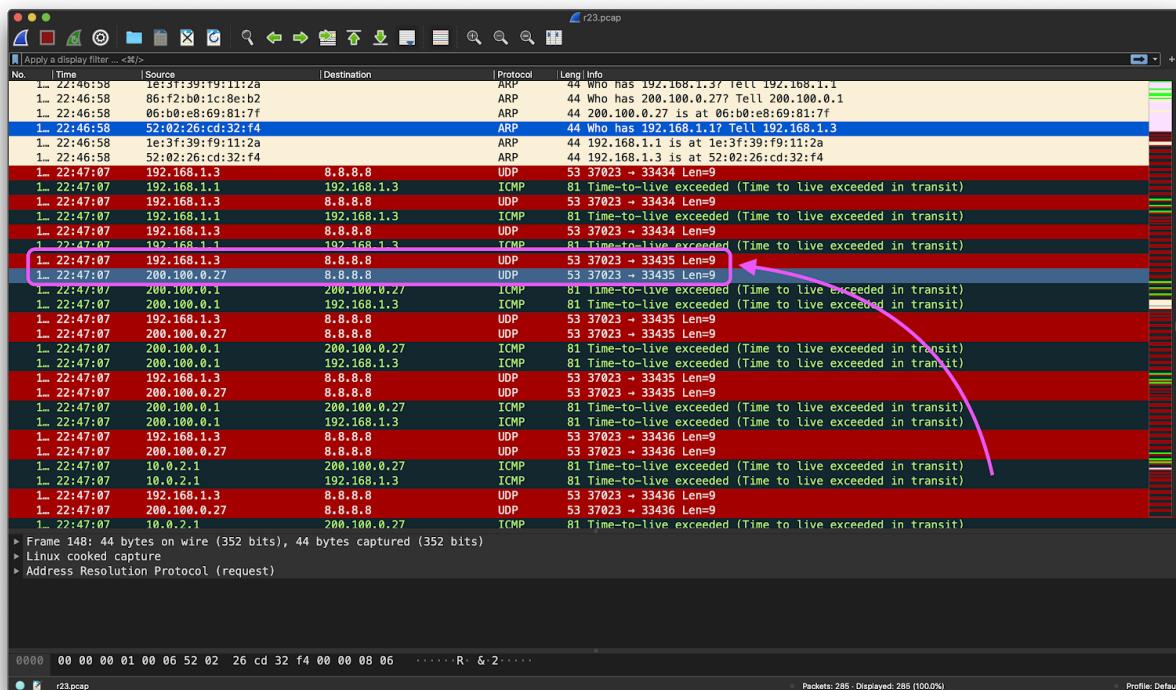
## DHCP

Para realizar las distintas pruebas que hicimos anteriormente desde la **pc24** y **pc22**, se necesitaba que ambas PCs tuvieran asignada una dirección IP, ya que los comandos funcionaron esto nos indica que efectivamente la configuración de DHCP se realizó correctamente. Además si ejecutamos el comando **ifconfig** desde las pcs, podemos verificar la IP que fue asignada a cada pc. Haremos un análisis más profundo sobre los mensajes DHCP que se enviaron en la siguiente sección.

## Análisis

### NAT

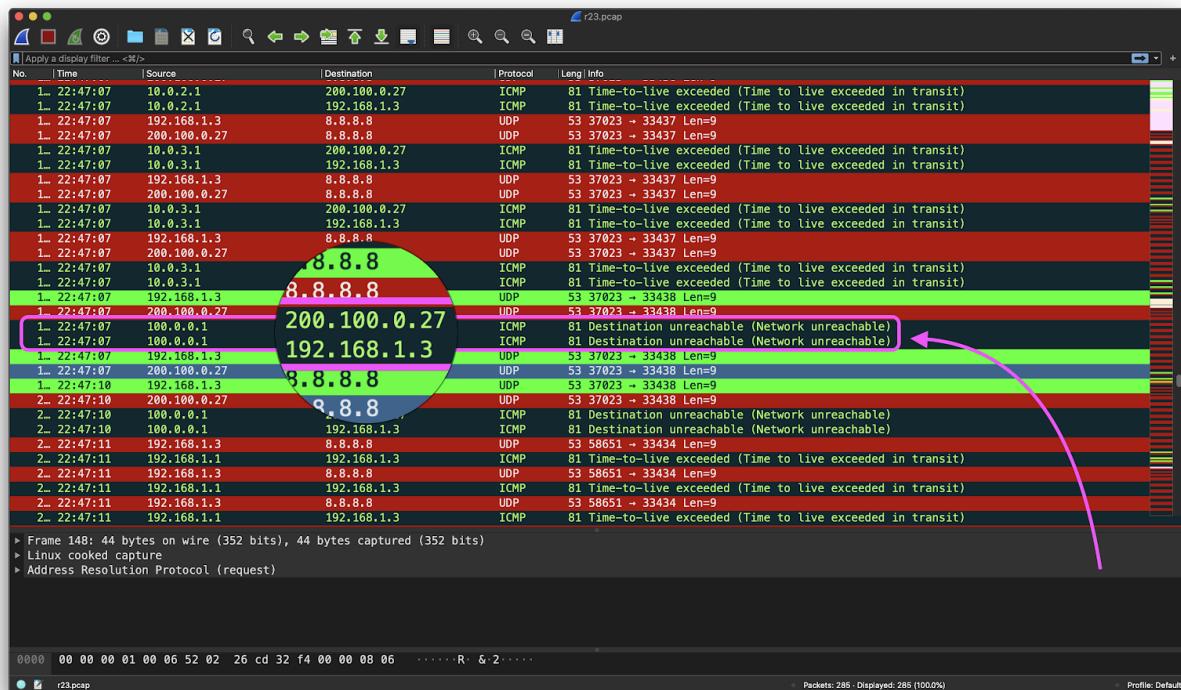
Para verificar el funcionamiento de NAT desde la **pc24** podemos observar la siguiente captura:



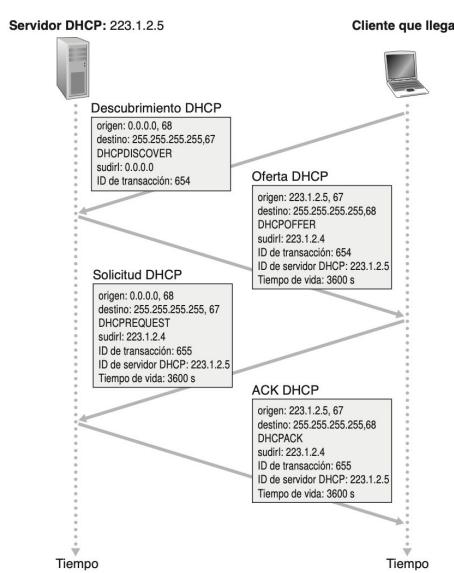
Aquí podemos observar que los primeros tres paquetes ICMP echo reply que vuelven a **pc24** vienen directamente desde el router **r23** ya que son descartados ahí debido al TTL igual a 1, por esta razón solo podemos ver direcciones IP privadas, las de la LAN2.

Como se puede observar en los paquetes resaltados, el primer paquete con TTL = 2 saldrá del router **r23**. Por lo tanto se espera que NAT cambie la dirección IP privada del paquete por la dirección pública del router **r23**. A este le llega un paquete UDP el cual tiene que enviar a **8.8.8.8**, por lo cual **r23** crea un nuevo paquete con su dirección pública, en este caso **200.100.0.27**, y con el mismo destino.

En la siguiente captura se puede observar la traducción que hace NAT una vez llega desde el ISP un paquete ICMP echo reply, tiene que crear otro paquete con destino a **pc24** (dirección IP privada de **pc24**) y para hacer esto tiene que hacer uso de la tabla de traducciones NAT.



## DHCP

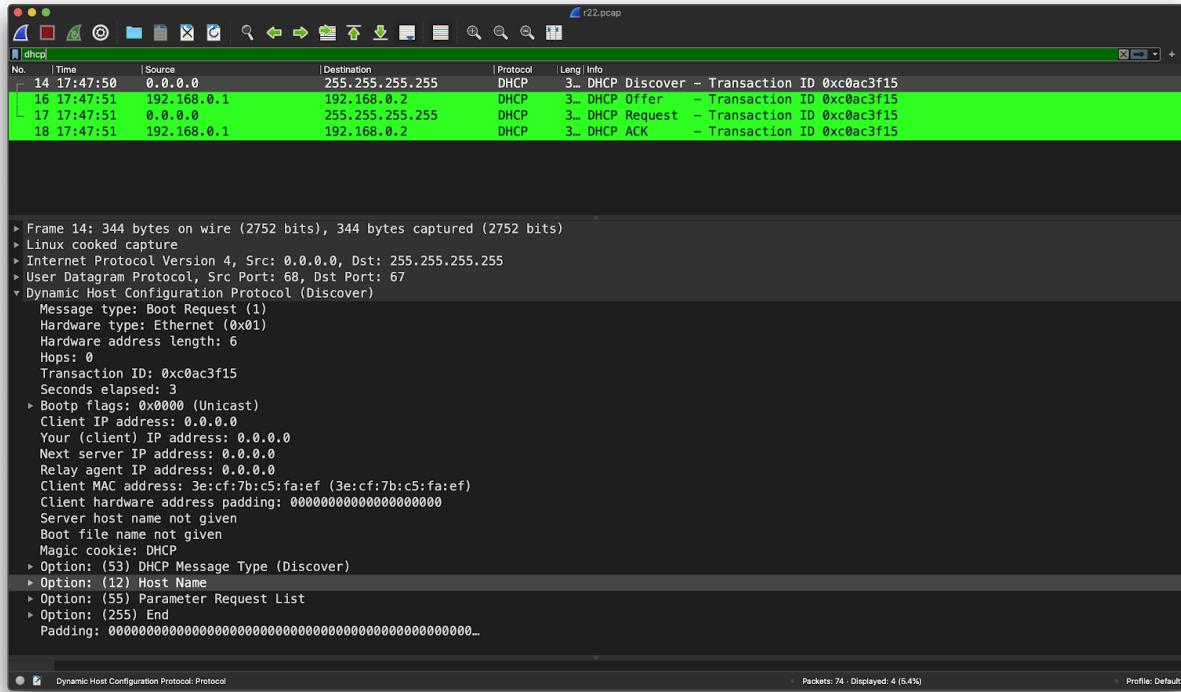
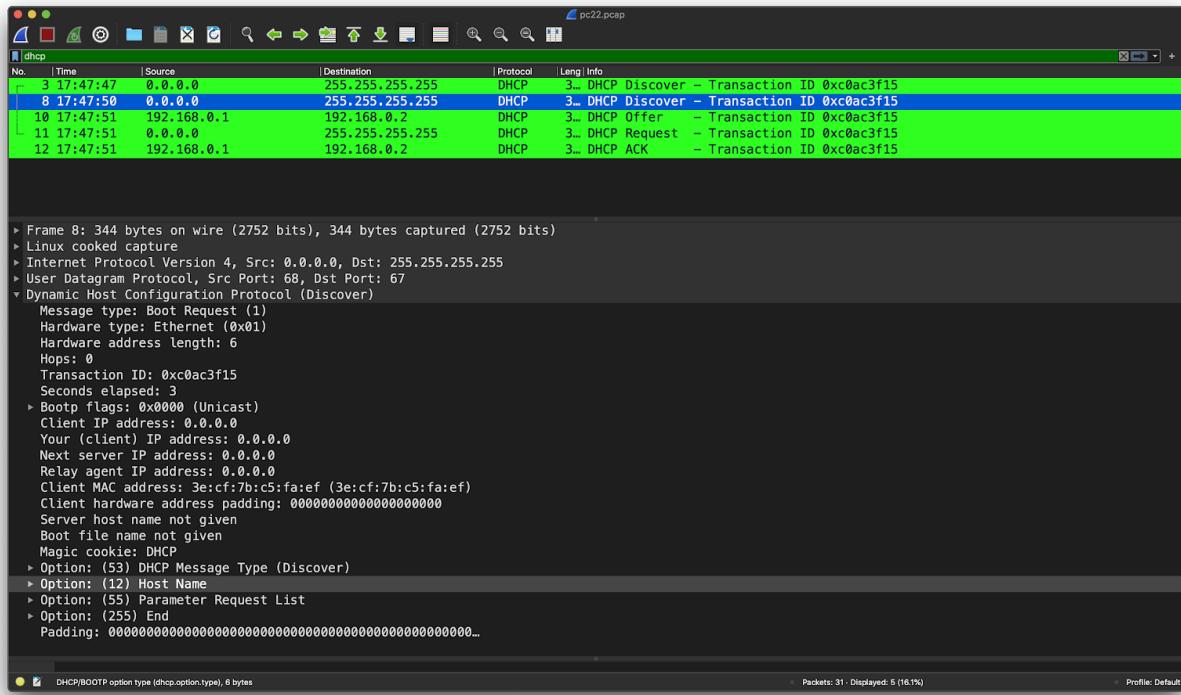


Para analizar los mensajes DHCP intercambiados observemos por ejemplo las capturas que realizamos de la **pc22** y del router **r22**.

Para que la **pc22** logre obtener una IP, primero debe mandar un mensaje **DHCP Discover**, este mensaje se manda a la dirección de Broadcast al puerto **68**, este mensaje será recibido por el servidor **DHCP** que se encuentra en **r22**, luego **r22** envía un mensaje **DHCP Offer**. Vemos aquí que este mensaje es enviado en Unicast a la dirección **192.168.0.2** siendo ofrecida. En este caso notamos que difiere de lo presentado en el libro (como puede observarse en la imagen a la izquierda<sup>19</sup>), aunque en el RFC no indica que esto sea un MUST. La

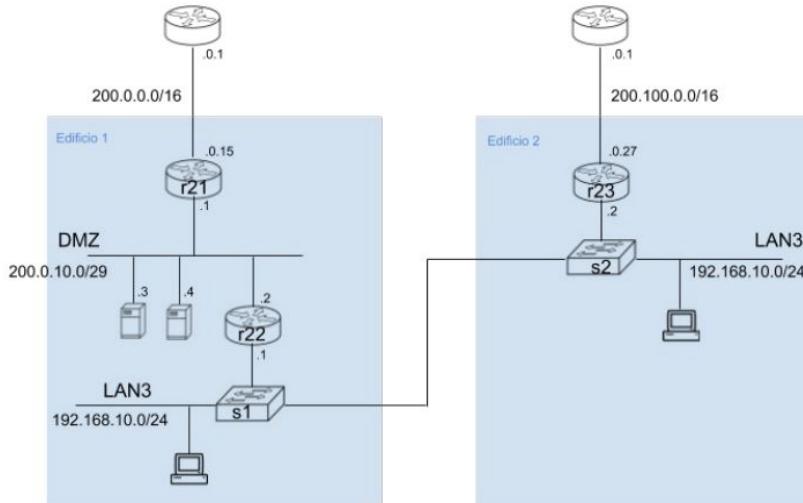
<sup>19</sup> Kurose, Ross - Computer Networking A Top-Down Approach: 7th Edition

respuesta de este Offer, el **DHCP Request**, es enviada por el **pc22** en Broadcast. El servidor le responde en Unicast a la **pc22** con un paquete **DHCP ACK**, finalizando este proceso.



## Parte 3: Institución con Switches

Como parte de una reestructura, la Institución re-diseña su topología de acuerdo a la siguiente figura.



Los nuevos requerimientos para el diseño de la red son:

- Las oficinas de ambos edificios deben estar en una única LAN3.
- Todo el tráfico hacia y desde Internet de la LAN3 debe circular por r23, excepto el tráfico hacia y desde la DMZ que debe circular por r22.
- r21 queda dedicado a servir las necesidades de Internet de los servidores de la DMZ.

### Implementación en la Red

#### Configuración

Esta parte se realizó a partir de lo configurado en el Entregable 2.2. Es decir, que contiene OSPF. Sin embargo, como la topología de los edificios ha cambiado, se realizaron ajustes tanto a DHCP como a NAT.

DHCP ahora está configurado en r23, y es el encargado de asignar IPs dinámicas a toda la nueva LAN3. La configuración nueva de DHCP es:

```
# static route and default route
option rfc3442-classless-static-routes code 121 = array of integer 8;
option rfc3442-classless-static-routes 0, 192,168,10,2,
                                29, 200,0,10,0, 192,168,10,1;
option ms-classless-static-routes code 249 = array of integer 8;
option ms-classless-static-routes 0, 192,168,10,2,
                                29, 200,0,10,0, 192,168,10,1;
```

La ruta por defecto en este caso es **r23** (192.168.10.2), ya que por requerimiento de letra el tráfico hacia el ISP y el internet deberá pasar por este router.

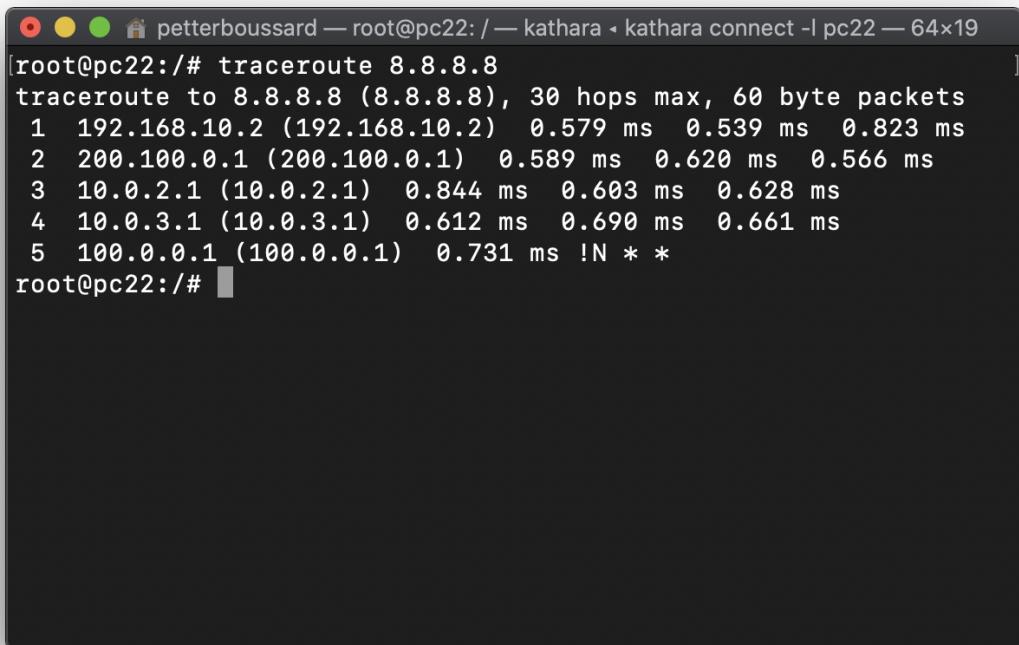
Además, se configuró para que las rutas hacia la **DMZ** (200.0.10.0/29) pasen por **r22** (192.168.10.1).

Para configuración de NAT tanto r22 como r23 se encargan de cambiar las IPs al salir de la LAN3. Se mantiene análogo a partes anteriores de la tarea.

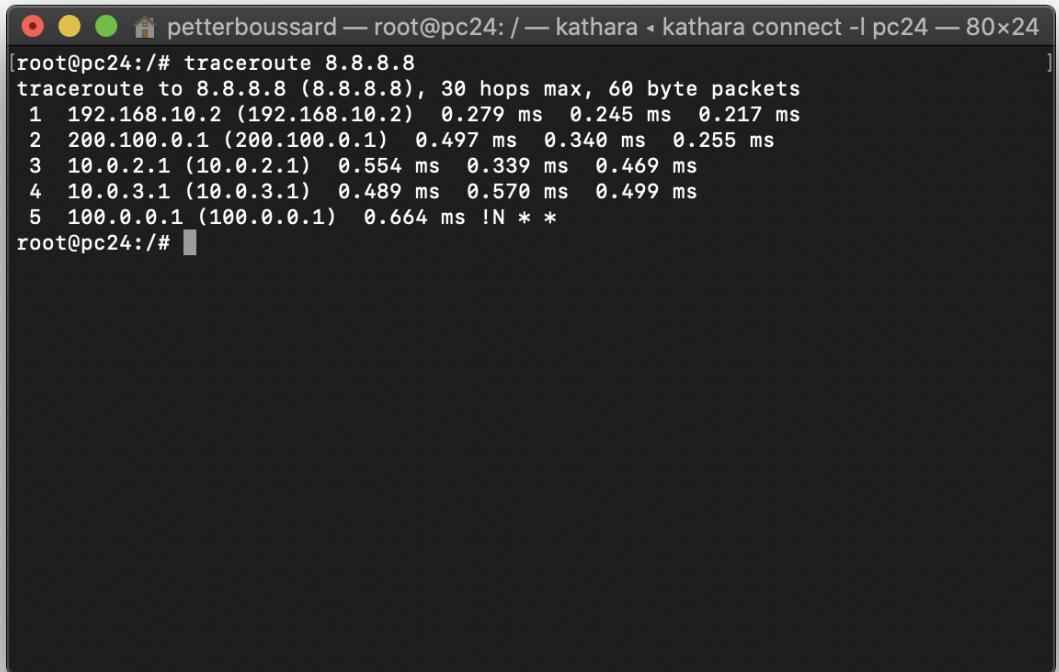
#### Tráfico de LAN3 a Internet

Todos los paquetes con destinos desconocidos, es decir, con IPs que no son ninguna de las subredes manejadas en este laboratorio, los enviamos al router r3, por lo que al querer enviar paquetes a internet deberían llegar a ese router.

En la siguiente imagen se ve una ejecución de traceroute hacia un servidor de google (8.8.8.8) desde la pc22 de la LAN3. Vemos como el primer router del camino es 192.168.10.2 (router r23), luego 200.100.0.1 (router r13), luego 10.0.2.1 (router r14), luego 10.0.3.1 (router r11) y por último 100.0.0.1 (router r3).

A screenshot of a terminal window titled "petterboussard — root@pc22: / — kathara - kathara connect -l pc22 — 64x19". The window shows the command "traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets" followed by a list of routers: 1 192.168.10.2 (192.168.10.2) 0.579 ms 0.539 ms 0.823 ms, 2 200.100.0.1 (200.100.0.1) 0.589 ms 0.620 ms 0.566 ms, 3 10.0.2.1 (10.0.2.1) 0.844 ms 0.603 ms 0.628 ms, 4 10.0.3.1 (10.0.3.1) 0.612 ms 0.690 ms 0.661 ms, 5 100.0.0.1 (100.0.0.1) 0.731 ms !N \* \*. The terminal prompt is "root@pc22:/#".

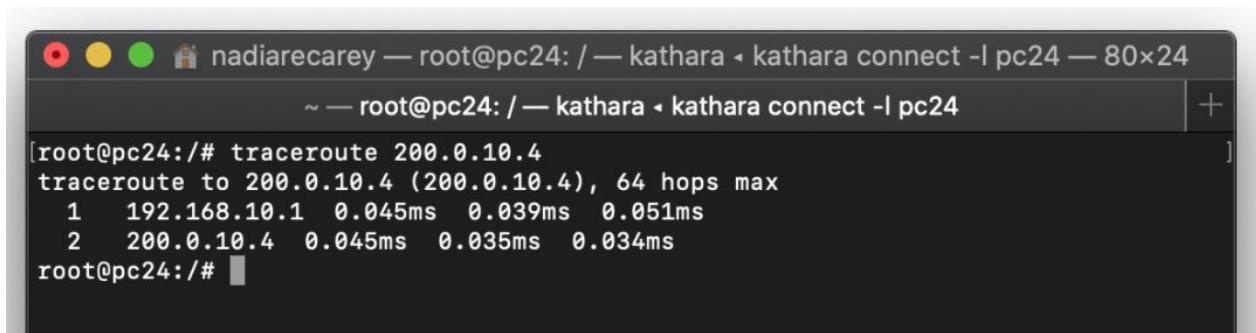
```
[root@pc22:/# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  192.168.10.2 (192.168.10.2)  0.579 ms  0.539 ms  0.823 ms
 2  200.100.0.1 (200.100.0.1)  0.589 ms  0.620 ms  0.566 ms
 3  10.0.2.1 (10.0.2.1)  0.844 ms  0.603 ms  0.628 ms
 4  10.0.3.1 (10.0.3.1)  0.612 ms  0.690 ms  0.661 ms
 5  100.0.0.1 (100.0.0.1)  0.731 ms  !N * *
```



```
[root@pc24:/# traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
1 192.168.10.2 (192.168.10.2) 0.279 ms 0.245 ms 0.217 ms
2 200.100.0.1 (200.100.0.1) 0.497 ms 0.340 ms 0.255 ms
3 10.0.2.1 (10.0.2.1) 0.554 ms 0.339 ms 0.469 ms
4 10.0.3.1 (10.0.3.1) 0.489 ms 0.570 ms 0.499 ms
5 100.0.0.1 (100.0.0.1) 0.664 ms !N * *
root@pc24:/#
```

### Tráfico de LAN3 a DMZ

En las siguientes capturas se ve el traceroute realizado desde la pc24 de la LAN hacia el servidor 200.0.10.4 dentro de la DMZ, y luego e traceroute desde la pc22 hacia el mismo servidor. Ambas envian sus paquetes al router r22 (192.168.10.1) y éste reenvía los mismos hacia el servidor ya que estan directamente conectados.



```
[root@pc24:/# traceroute 200.0.10.4
traceroute to 200.0.10.4 (200.0.10.4), 64 hops max
1 192.168.10.1 0.045ms 0.039ms 0.051ms
2 200.0.10.4 0.045ms 0.035ms 0.034ms
root@pc24:/#
```

```

nadiarecarey — root@pc22: / — kathara ↵ kathara connect -l pc22 — 80x24
~ — root@pc22: / — kathara ↵ kathara connect -l pc22

[root@pc22:/# traceroute 200.0.10.4
traceroute to 200.0.10.4 (200.0.10.4), 64 hops max
 1  192.168.10.1  0.032ms  0.042ms  0.050ms
 2  200.0.10.4  0.034ms  0.033ms  0.114ms
root@pc22:/#

```

## Tráfico de Internet a la DMZ

Como se ve en la captura, ejecutando traceroute desde el router r3 (que consideramos que es la salida a internet) hacia uno de los servidores de la DMZ, vemos que el camino es primero 100.0.100.50 (el router r11 del backbone), luego 10.0.3.2 (router r14), luego 200.0.0.15 (router r21) y por último el router r21 envía el paquete al servidor ya que están directamente conectados.

```

petterboussard — root@r3: / — kathara ↵ kathara connect -l r3 — 70x9
[root@r3:/# traceroute 200.0.10.3
traceroute to 200.0.10.3 (200.0.10.3), 30 hops max, 60 byte packets
 1  100.0.100.50 (100.0.100.50)  0.295 ms  0.160 ms  0.104 ms
 2  10.0.3.2 (10.0.3.2)  0.294 ms  0.196 ms  0.208 ms
 3  200.0.0.15 (200.0.0.15)  0.434 ms  0.250 ms  0.439 ms
 4  200.0.10.3 (200.0.10.3)  0.508 ms  0.568 ms  0.561 ms
root@r3:/#

```

## Estudio Comparativo

### Comparación Partes 2 y 3

Un dominio de broadcast es un área de una red de computadoras, formada por todas las computadoras y dispositivos de red que se pueden alcanzar enviando una trama a la dirección de broadcast de la capa de enlace de datos<sup>20</sup>.

Un dominio de colisión es un segmento **físico** de una red de computadores donde es posible que las tramas puedan "colisionar" (interferir) con otras.<sup>21</sup>

Por lo tanto, en la parte 2 tenemos 2 LANs, cada una con dominios de colisión y dominios de broadcast individuales. En este caso, ya que en las LANs no hay switches, estos dos dominios

---

<sup>20</sup> [Wikipedia - Dominio de broadcast](#)

<sup>21</sup> [Wikipedia - Dominio de colisión](#)

coinciden. En cambio en la parte 3, tenemos una única LAN con un único dominio de broadcast (la dirección broadcast casi siempre es la última IP del rango de IPs de la subred), pero varios dominios de colisión, separados por los switches.

Para ver que los dominios de broadcast son diferentes en ambas partes, bastaría con enviar un mensaje en broadcast desde alguna de las pcs. En el caso de la parte 2 los mensajes broadcast enviados, por ejemplo, desde la pc24 no deberían llegar a la pc22 ya que no están dentro del dominio de broadcast (ni la misma LAN). En cambio en la parte 3, si bien no están dentro del mismo dominio de colisión, un mensaje enviado en broadcast debería llegarle a todos los dispositivos conectados a la red, incluso, ya que los switches en estos casos funcionan como hubs.

A modo de ejemplo en las siguientes capturas de la parte 3 podremos ver como un ping a broadcast desde **pc24** llega tanto a **pc22** como a **s1** y **s2**.<sup>22</sup>

```
nadiarecarey — root@pc24: / — kathara -> kathara connect -l pc24 — 80x24
~ — root@pc24: / — kathara -> kathara connect -l pc24

For info, please visit https://www.isc.org/software/dhcp

Listening on LPF/eth0/b6:35:c9:dd:dd:19
Sending on  LPF/eth0/b6:35:c9:dd:dd:19
Sending on  Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 8
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 9
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 11
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 11
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 9
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 7
DHCPREQUEST of 192.168.10.3 on eth0 to 255.255.255.255 port 67
DHCPoffer of 192.168.10.3 from 192.168.10.2
DHCPACK of 192.168.10.3 from 192.168.10.2
bound to 192.168.10.3 -- renewal in 17167 seconds.
done.

--- End Startup Commands Log

[root@pc24:/# ping 255.255.255.255
Do you want to ping broadcast? Then -b. If not, check your local firewall rules.
[root@pc24:/# ping 255.255.255.255 -b
WARNING: pinging broadcast address
PING 255.255.255.255 (255.255.255.255) 56(84) bytes of data.
```

<sup>22</sup> Se tomaron estas máquinas como muestra; también deberían de llegar a r22 y r23.

```

nadiarecarey — root@s1: / — kathara ✧ kathara connect -l s1 — 106x24
~ — root@s1: / — kathara ✧ kathara connect -l s1

01:22:56.223758 STP 802.1d, Config, Flags [none], bridge-id 8000.0e:e1:af:84:56:28.8002, length 35
01:22:56.223783 STP 802.1d, Config, Flags [none], bridge-id 8000.0e:e1:af:84:56:28.8001, length 35
01:22:57.184051 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 47, length 64
01:22:57.184252 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 47, length 64
01:22:57.184280 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 47, length 64
01:22:57.184051 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 47, length 64
01:22:57.247946 STP 802.1d, Config, Flags [none], bridge-id 8000.1a:37:25:0d:c4:71.8002, length 35
01:22:58.207716 STP 802.1d, Config, Flags [none], bridge-id 8000.0e:e1:af:84:56:28.8003, length 35
01:22:58.207756 STP 802.1d, Config, Flags [none], bridge-id 8000.0e:e1:af:84:56:28.8002, length 35
01:22:58.207777 STP 802.1d, Config, Flags [none], bridge-id 8000.0e:e1:af:84:56:28.8001, length 35
01:22:58.207984 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:58.208080 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:58.208136 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:58.207984 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:59.232742 STP 802.1d, Config, Flags [none], bridge-id 8000.1a:37:25:0d:c4:71.8002, length 35
01:22:59.233130 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
01:22:59.233189 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
01:22:59.233213 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
01:22:59.233130 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
^C
582 packets captured
665 packets received by filter
69 packets dropped by kernel
root@s1:/#

```

```

nadiarecarey — root@s2: / — kathara ✧ kathara connect -l s2 — 110x24
~ — root@s2: / — kathara ✧ kathara connect -l s2

01:22:57.247857 STP 802.1d, Config, Flags [none], bridge-id 8000.1a:37:25:0d:c4:71.8002, length 35
01:22:57.247879 STP 802.1d, Config, Flags [none], bridge-id 8000.1a:37:25:0d:c4:71.8001, length 35
01:22:58.207823 STP 802.1d, Config, Flags [none], bridge-id 8000.0e:e1:af:84:56:28.8003, length 35
01:22:58.207877 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:58.207920 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:58.207940 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:58.207877 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 48, length 64
01:22:59.232537 STP 802.1d, Config, Flags [none], bridge-id 8000.1a:37:25:0d:c4:71.8003, length 35
01:22:59.232583 STP 802.1d, Config, Flags [none], bridge-id 8000.1a:37:25:0d:c4:71.8002, length 35
01:22:59.232638 STP 802.1d, Config, Flags [none], bridge-id 8000.1a:37:25:0d:c4:71.8001, length 35
01:22:59.232973 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
01:22:59.233066 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
01:22:59.233086 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
01:22:59.232973 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 49, length 64
01:23:00.257445 STP 802.1d, Config, Flags [none], bridge-id 8000.0e:e1:af:84:56:28.8003, length 35
01:23:00.257953 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 50, length 64
01:23:00.257989 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 50, length 64
01:23:00.258055 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 50, length 64
01:23:00.257953 IP 192.168.10.3 > 255.255.255.255: ICMP echo request, id 135, seq 50, length 64
^C
575 packets captured
652 packets received by filter
63 packets dropped by kernel
root@s2:/#

```

Sin embargo, el mismo experimento en la Parte 2, arroja los siguientes resultados:

```
nadiarecarey — root@pc24: / — kathara ✧ kathara connect -l pc24 — 80x24
~ — root@pc24: / — kathara ✧ kathara connect -l pc24

[root@pc24:/# ping 255.255.255.255 -b
WARNING: pinging broadcast address
PING 255.255.255.255 (255.255.255.255) 56(84) bytes of data.
^C
--- 255.255.255.255 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5078ms

root@pc24:/#
```

```
nadiarecarey — root@r23: / — kathara ✧ kathara connect -l r23 — 130x24
~ — root@r23: / — kathara ✧ kathara connect -l r23

root@r23:/# tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
[listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
01:39:29.308221 IP 192.168.1.3 > 255.255.255.255: ICMP echo request, id 138, seq 1, length 64
01:39:29.309289 IP localhost.33694 > localhost.domain: 52001+ PTR? 255.255.255.in-addr.arpa. (46)
01:39:29.309331 IP localhost > localhost: ICMP localhost udp port domain unreachable, length 82
01:39:29.309486 IP localhost.56239 > localhost.domain: 52001+ PTR? 255.255.255.in-addr.arpa. (46)
01:39:30.325927 IP 192.168.1.3 > 255.255.255.255: ICMP echo request, id 138, seq 2, length 64
01:39:31.328494 IP 192.168.1.3 > 255.255.255.255: ICMP echo request, id 138, seq 3, length 64
01:39:32.339585 IP 192.168.1.3 > 255.255.255.255: ICMP echo request, id 138, seq 4, length 64
01:39:33.362776 IP 192.168.1.3 > 255.255.255.255: ICMP echo request, id 138, seq 5, length 64
01:39:34.386800 IP 192.168.1.3 > 255.255.255.255: ICMP echo request, id 138, seq 6, length 64
^C
9 packets captured
22 packets received by filter
10 packets dropped by kernel
root@r23:/#
```

```

root@pc22:/# tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel
root@pc22:#

```

En definitiva, el mensaje enviado en broadcast desde la pc24 no llega a la pc22.

## Self-learning

En el contexto de switches, self-learning hace referencia a que la **tabla de conmutación** de estos se construye de forma automática, dinámica y autónoma, sin requerir intervención del administrador de red ni protocolos de configuración.

La **tabla de conmutación** de un switch almacena (1) la dirección MAC de un host que envió un paquete (dirección MAC de origen) y el mismo “pasó” por el switch, (2) La interfaz del switch que lleva hacia dicha dirección MAC y (3) instante en el que la entrada fue incluida en la tabla.

Funciones de un switch (Filtrado y Reenvío)

El **filtrado** es la función del switch que determina si una trama debe ser reenviada a alguna interfaz o debe ser descartada. El **reenvío** es la función del switch que determina las interfaces a las que una trama debe dirigirse y luego envía la trama a esas interfaces. Las funciones de filtrado y reenvío del switch se realizan utilizando la **tabla de conmutación**.<sup>23</sup>

Cuando una trama llega a determinada interfaz de un switch, el mismo observa la dirección MAC hacia dónde se dirige y consulta la tabla de conmutación para saber a cuál interfaz debe enviar la trama, en este caso, existen 3 escenarios posibles:

- No hay entrada en la tabla de conmutación para la dirección MAC de destino, en este caso el switch reenvía copias de la trama a todas las interfaces.

---

<sup>23</sup> Kurose, Ross - Computer Networking A Top-Down Approach: 7th Edition

- La dirección MAC de destino de la trama se encuentra en la misma interfaz por donde llegó, en este caso, al no ser necesario reenviar la trama a las demás interfaces, la misma es destacada.
- Existe una entrada en la tabla de conmutación que indica hacia qué interfaz del switch dirigir la trama y esta no es la misma interfaz por donde llegó, en este caso lo que hace en switch es reenviar la trama a la interfaz correspondiente.

## Prueba de Self-learning

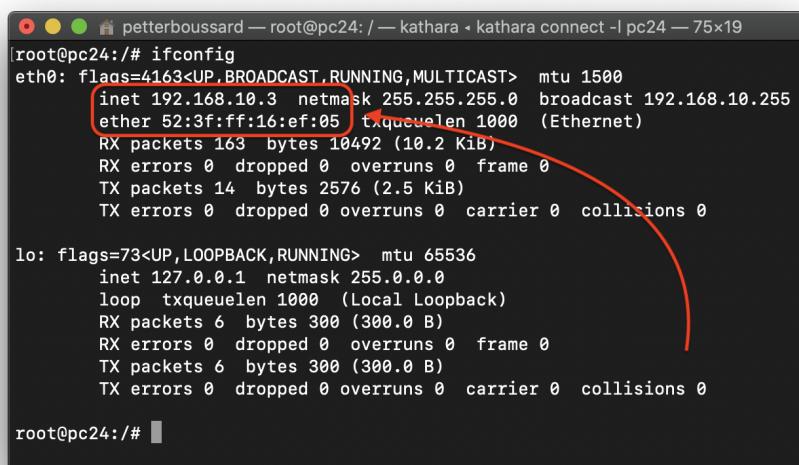
Para realizar la prueba sobre self-learning decidimos realizar un ping desde **pc24** hasta **pc22** y verificar que el primer paquete **ICMP echo request** es recibido tanto por **r23**, **r22** y la **pc22**, pero los demás ya no son recibidos en estos routers ya que los switches ya tendrán almacenadas las direcciones MAC de ambos pcs en sus respectivas tablas de commutaciones.

Lo primero que hicimos fue hacer un ping desde **pc24** hasta **pc22** con el objetivo de que **pc24** envíe un mensaje **ARP** para aprender la dirección MAC de **pc22**.

Luego, configuramos el **ageing time** de las entradas de las tablas de los commutaciones de los switches en 10 segundos para que de esta manera se eliminen las direcciones MAC de **pc22** y **pc24**.

Después de hacer esta configuración volvemos a realizar el ping (esta vez no se enviará un mensaje **ARP** porque **pc24** ya sabe la dirección MAC de **pc22**) y logramos observar que el primer paquete **ICMP echo request** llega solo una vez a los routers **r23** y **r22** pero los demás no. Esto sucede ya que la primera vez que un paquete echo request se envía, este pasa por los switches pero como estos no conocen la dirección MAC reenvían este paquete a todas sus interfaces (esto sucede para ambos switches al recibir el primer paquete ICMP), pero luego de que **pc22** envía su paquete **ICMP echo reply**, los switches pueden observar que va dirigido hacia **pc24** (y ya conocen la dirección MAC), por lo tanto no reenvían estos paquetes a las interfaces que se dirigen hacia los routers.

Observemos las direcciones IP y MAC de **pc24** y **pc22** verificando que no se encuentren en las entradas de **s1** y **s2**:



```
[root@pc24:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.10.3  netmask 255.255.255.0  broadcast 192.168.10.255
              ether 52:3f:ff:16:ef:05  txqueuelen 1000  (Ethernet)
                    RX packets 163  bytes 10492 (10.2 Kib)
                    RX errors 0  dropped 0  overruns 0  frame 0
                    TX packets 14  bytes 2576 (2.5 Kib)
                    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
          loop  txqueuelen 1000  (Local Loopback)
            RX packets 6  bytes 300 (300.0 B)
            RX errors 0  dropped 0  overruns 0  frame 0
            TX packets 6  bytes 300 (300.0 B)
            TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

root@pc24:/# ]
```

```
[root@pc22:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.10.4 netmask 255.255.255.0 broadcast 192.168.10.255
        ether ca:35:57:d6:9c:b0 txqueuelen 1000 (Ethernet)
        RX packets 181 bytes 11536 (11.2 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 14 bytes 2576 (2.5 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        loop txqueuelen 1000 (Local Loopback)
        RX packets 6 bytes 300 (300.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 6 bytes 300 (300.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

root@pc22:/# ]
```

```
[root@s1:/# brctl showmacs br0
port no mac addr           is local?      ageing timer
  3   5e:a0:25:46:41:1b    yes          0.00
  3   5e:a0:25:46:41:1b    yes          0.00
  3   7e:81:13:7e:2f:13   no           0.05
  1   9a:ff:48:3d:bc:6c   yes          0.00
  1   9a:ff:48:3d:bc:6c   yes          0.00
  2   d6:70:d5:3e:a9:6e   yes          0.00
  2   d6:70:d5:3e:a9:6e   yes          0.00
root@s1:/# ]
```

```
[root@s2:/# brctl showmacs br0
port no mac addr           is local?      ageing timer
  2   5e:a0:25:46:41:1b    no           1.85
  2   7e:81:13:7e:2f:13   yes          0.00
  2   7e:81:13:7e:2f:13   yes          0.00
  1   b2:6b:0b:04:db:3e   yes          0.00
  1   b2:6b:0b:04:db:3e   yes          0.00
  3   d2:96:5d:50:4c:00   yes          0.00
  3   d2:96:5d:50:4c:00   yes          0.00
root@s2:/# ]
```

Luego de ejecutar el ping desde **pc24** hasta **pc22**, podemos ver que las direcciones MAC de ambos se encuentran en las tablas de conmutación de ambos switches y además podemos ver que al capturar el tráfico generado en **r22** y **r23** podemos ver sólo el primer paquete **ICMP echo request**

```
[root@pc24:/# ping 192.168.10.4
PING 192.168.10.4 (192.168.10.4) 56(84) bytes of data.
64 bytes from 192.168.10.4: icmp_seq=1 ttl=64 time=1.44 ms
64 bytes from 192.168.10.4: icmp_seq=2 ttl=64 time=0.859 ms
64 bytes from 192.168.10.4: icmp_seq=3 ttl=64 time=0.425 ms
64 bytes from 192.168.10.4: icmp_seq=4 ttl=64 time=0.350 ms
64 bytes from 192.168.10.4: icmp_seq=5 ttl=64 time=0.358 ms
64 bytes from 192.168.10.4: icmp_seq=6 ttl=64 time=1.03 ms
64 bytes from 192.168.10.4: icmp_seq=7 ttl=64 time=0.553 ms
64 bytes from 192.168.10.4: icmp_seq=8 ttl=64 time=0.424 ms
^C
--- 192.168.10.4 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7155ms
rtt min/avg/max/mdev = 0.350/0.681/1.448/0.372 ms
root@pc24:/# ]
```

```
[root@s1:/# brctl showmacs br0
port no mac addr          is local?      ageing timer
 3    2a:c1:ee:10:c1:76    no            26.32
 1    2a:f5:c8:e9:cb:82    no            1.75
 3    52:3f:ff:16:ef:05    no            2.77
 3    5e:a0:25:46:41:1b    yes           0.00
 3    5e:a0:25:46:41:1b    yes           0.00
 3    7e:81:13:7e:2f:13    no            0.02
 1    82:66:7c:d5:89:84    no            22.23
 1    9a:ff:48:3d:bc:6c    yes           0.00
 1    9a:ff:48:3d:bc:6c    yes           0.00
 2    ca:35:57:d6:9c:b0    no            2.77
 2    d6:70:d5:3e:a9:6e    yes           0.00
 2    d6:70:d5:3e:a9:6e    yes           0.00
root@s1:/# ]
```

```

petterboussard — root@s2: / — kathara ✧ kathara connect -l s2 — 62x16
[root@s2:/# brctl showmacs br0
port no mac addr          is local?    ageing timer
  3    2a:c1:ee:10:c1:76   no           21.82
  2    2a:f5:c8:e9:cb:82   no           25.91
  3    52:3f:ff:16:ef:05   no           26.94
  2    5e:a0:25:46:41:1b   no           0.18
  2    7e:81:13:7e:2f:13   yes          0.00
  2    7e:81:13:7e:2f:13   yes          0.00
  2    9e:38:76:c5:c5:4c   no           13.62
  1    b2:6b:0b:04:db:3e   yes          0.00
  1    b2:6b:0b:04:db:3e   yes          0.00
  1    ba:73:60:7f:b0:ce   no           21.82
  2    ca:35:57:d6:9c:b0   no           26.94
  3    d2:96:5d:50:4c:00   yes          0.00
  3    d2:96:5d:50:4c:00   yes          0.00
root@s2:/# ]

```

```

petterboussard — root@r23: / — kathara ✧ kathara connect -l r23 — 101x15
[root@r23:/# tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
14:42:52.999954 STP 802.1d, Config, Flags [none], bridge-id 8000.7e:81:13:7e:2f:13.8001, length 35
14:42:53.487779 IP 192.168.10.3 > 192.168.10.4: ICMP echo request, id 140, seq 1, length 64
14:42:53.510298 IP localhost.51124 > localhost.domain: 37644+ PTR? 4.10.168.192.in-addr.arpa. (43)
14:42:53.510375 IP localhost > localhost: ICMP localhost udp port domain unreachable, length 79
14:42:53.510543 IP localhost.35523 > localhost.domain: 37644+ PTR? 4.10.168.192.in-addr.arpa. (43)
14:42:54.983854 STP 802.1d, Config, Flags [none], bridge-id 8000.7e:81:13:7e:2f:13.8001, length 35
14:42:56.934515 STP 802.1d, Config, Flags [none], bridge-id 8000.7e:81:13:7e:2f:13.8001, length 35
14:42:58.918560 STP 802.1d, Config, Flags [none], bridge-id 8000.7e:81:13:7e:2f:13.8001, length 35
14:43:00.966771 STP 802.1d, Config, Flags [none], bridge-id 8000.7e:81:13:7e:2f:13.8001, length 35
^C
9 packets captured
22 packets received by filter

```

```

petterboussard — root@r22: / — kathara ✧ kathara connect -l r22 — 101x17
[root@r22:/# tcpdump -i any
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes
14:42:52.999899 STP 802.1d, Config, Flags [none], bridge-id 8000.5e:a0:25:46:41:1b.8001, length 35
14:42:53.488013 IP 192.168.10.3 > 192.168.10.4: ICMP echo request, id 140, seq 1, length 64
14:42:53.493915 IP localhost.43140 > localhost.domain: 33641+ PTR? 4.10.168.192.in-addr.arpa. (43)
14:42:53.494207 IP localhost > localhost: ICMP localhost udp port domain unreachable, length 79
14:42:53.494340 IP localhost.37628 > localhost.domain: 33641+ PTR? 4.10.168.192.in-addr.arpa. (43)
14:42:54.983908 STP 802.1d, Config, Flags [none], bridge-id 8000.5e:a0:25:46:41:1b.8001, length 35
14:42:56.934452 STP 802.1d, Config, Flags [none], bridge-id 8000.5e:a0:25:46:41:1b.8001, length 35
14:42:58.918665 STP 802.1d, Config, Flags [none], bridge-id 8000.5e:a0:25:46:41:1b.8001, length 35
14:43:00.966600 STP 802.1d, Config, Flags [none], bridge-id 8000.5e:a0:25:46:41:1b.8001, length 35
^C
9 packets captured
22 packets received by filter
10 packets dropped by kernel
root@r22:/# ]

```

Con esta prueba podemos ver en acción el funcionamiento de **self-learning** que tienen los switches.