

UNIVERSIDAD NACIONAL DE CÓRDOBA
Facultad de Ciencias Exactas, Físicas y Naturales



Redes de Computadoras

Práctico 4: Ruteo Dinámico

Contrera, Ivan
Malano, Leandro

Host (router o PC)	red [nro red - direcc red]	Dirección IP
--------------------	----------------------------	--------------

r1	2 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::2
	5 - 2001:aaaa:bbbb:5::/64	2001:aaaa:bbbb:5::2
	6 - 2001:aaaa:bbbb:6::/64	2001:aaaa:bbbb:6::3
	D - 2001:aaaa:dddd:1::/64	2001:aaaa:dddd:1::3
r2	1 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::3
	2 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::3
	4 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::3
r3	6 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::2
r4	3 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::3
	4 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::2
	5 - 2001:aaaa:bbbb:2::/64	2001:aaaa:bbbb:2::2
r5	8 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::2
	11 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::2
	12 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::3
	D - 2001:aaaa:dddd:2::/64	2001:aaaa:dddd:2::2
r6	7 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::3
	8 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::3
	10 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::3
r7	12 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::2
r8	9 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::3
	10 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::2
	11 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::3
h1	1 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::10
h2	3 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::20
h3	7 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::10
h4	9 - 2001:aaaa:cccc:2::/64	2001:aaaa:cccc:2::20

--	--	--

- 1) Utilizando la máquina virtual corriendo Ubuntu desktop 16.04, se instalan las herramientas necesarias para emular enrutadores en contenedores: Docker CE (permite crear los contenedores), Docker Compose (herramienta que facilita la creación de los contenedores), quagga (software que brinda características de enrutamiento, utilizando algoritmos como ospf, bgp, etc) y GIT (para acceder al repositorio de archivos necesarios para acceder al ejemplo de dos routers utilizando ospfv3). Los comandos utilizados son los siguientes:

**instalar docker:*

```
$ curl -fsSL get.docker.com -o get-docker.sh
```

```
$ sh get-docker.sh
```

**instalar docker compose 1.21.1:*

```
$ sudo curl -L
```

```
https://github.com/docker/compose/releases/download/1.21.1/docker-compose-`uname
```

```
-s`-`uname -m` -o /usr/local/bin/docker-compose
```

```
$ chmod +x /usr/local/bin/docker-compose
```

instalar git:

```
$ sudo apt-get update
```

```
$ sudo apt-get install git
```

instalar quagga:

```
$ sudo apt-get install quagga
```

- 2) A continuación, se va a clonar el repositorio brindado por el docente, mencionado anteriormente. Para este caso, se utiliza el siguiente comando:

```
$ git clone https://github.com/maticue/docker\_quagga.git.
```

- 3) No fue necesario configurar docker para que soporte IPv6.

- 4) Para iniciar el entorno de prueba, donde se crearán los dos contenedores (routers), se debe entrar a la carpeta ospf ubicada en la ruta /home/usuario/docker_quagga/ospf, y luego ejecutarse el comando:

```
$ docker-compose up
```

```
root@ivanovic-VirtualBox:/home/ivanovic/docker_quagga/ospf# docker-compose up
Creating ospf_r3_1 ... done
Starting ospf_r1_1 ... done
Starting ospf_r2_1 ... done
Attaching to ospf_r3_1, ospf_r1_1, ospf_r2_1
```

NOTA: ospf_r3_1 es un contenedor extra, creado con fines experimentales en este caso particular.

5) Observando el archivo docker-compose.yml, contiene distintas partes que se analizarán a continuación:

*Sección *services*: se declaran todos los contenedores que van a crearse (en ese caso, los dos routers según el archivo original). Contiene algunas subsecciones, como **build**, que crea una imagen de dicho contenedor que se va a poder instanciar las veces que sean necesarias. **volumes**, monta las configuraciones del archivo del primer campo, se reflejan en el archivo del segundo campo, y son de solo lectura, según el valor *ro* del tercer campo (los campos se separan con `:`):

volumes:

- `./volumes/quagga/r1/zebra.conf:/etc/quagga/zebra.conf:ro`
- `./volumes/quagga/r1/ospfd.conf:/etc/quagga/ospfd.conf:ro`
- `./volumes/quagga/r1/ospf6d.conf:/etc/quagga/ospf6d.conf:ro`
- `./volumes/supervisord.conf:/etc/supervisor/conf.d/supervisord.conf:ro`

Con esto, por ejemplo, se cargarán las configuraciones de ospfv3 de acuerdo al archivo ospf6d.conf, en un determinado contenedor. **image** hace referencia a qué imagen se usará para crear el contenedor. Una imagen contiene todo lo necesario para crear el contenedor particular. **ports**, sirve para comunicarse entre procesos. Cuando el contenedor se inicia (por ejemplo r1), se le asigna un número de puerto (10011), el cual va a escuchar el puerto del proceso zebra (2601). Si el usuario quiere acceder a las configuraciones de zebra, mediante telnet deberá accederse al puerto 10011 y docker hará la conexión con el puerto 2601. Lo mismo se puede hacer si se quiere acceder al proceso de ospf para IPv6. Una vez hecha la conexión, se podrá configurar el router casi con los mismos comandos que usan los de CISCO. En **networks**, se especifican las redes a las que va a pertenecer un determinado contenedor o router, y las ip (versión 4 y 6) de sus interfaces.

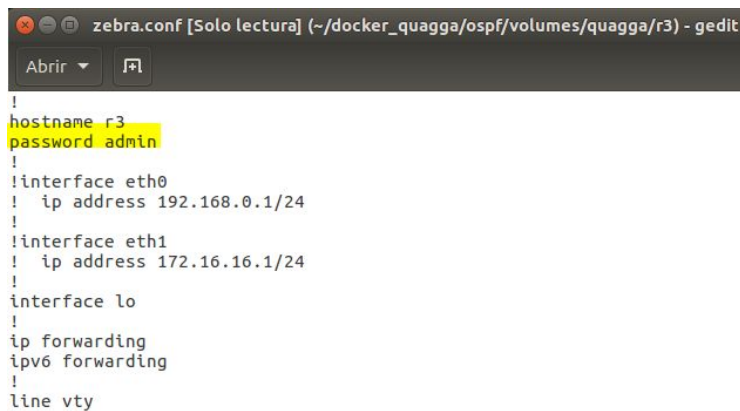
*Sección *networks*: crea las redes aisladas, independientes de los contenedores, disponibles para que estos las usen.

5.1) El servicio ospfv3 escucha en el puerto 2606, y para conectarse (desde r1), se hace con el comando:

```
$ telnet localhost 10013
```

6) A cada contenedor se les asocia archivos de configuración. Entre los mas destacados, estan los de ospfv3(ospf6d.conf) , ospf (ospfd.conf) y zebra (zebra.conf). El primero, habilita y configura el protocolo ospfv3, estableciendo costos, área, etc. El segundo, hace lo propio para ospf (IPv4), y el tercero, habilita los enrutamientos para IPv4/6.

6.1) El password para cada contenedor, se encuentra en el archivo zebra.conf:



```
!
hostname r3
password admin
!
!interface eth0
! ip address 192.168.0.1/24
!
!interface eth1
! ip address 172.16.16.1/24
!
interface lo
!
ip forwarding
ipv6 forwarding
!
line vty
```

7) Una vez hechas las pruebas anteriores, se modifica el archivo *docker-compose.yml*, agregando los routers restantes y los cuatro hosts. El archivo está disponible en <https://drive.google.com/open?id=1A-j5rPpVL8-x21iAT2wqmObtolljOeoS>.

8) Para que puedan cargarse las configuraciones de ospf, se crean los archivos necesarios (*ospf6d.conf* y *zebra.conf*) para cada router. Los hosts no necesitan, ya que tienen una ruta por defecto, por donde enviará cualquier paquete cuyo destino no sea una ip propia. Un ejemplo de cada archivo de r5, se muestran en las siguientes capturas:

ospf6d.conf

zebra.conf

```
!
hostname r5
password admin
!
!interface eth0
! ip address 192.168.0.1/24
!
!interface eth1
! ip address 172.16.16.1/24
!
interface lo
!
ipv6 forwarding
!
line vty
```

```

! *- ospfv3 *-
!
! OSPF6d configuration file
!
!
hostname r1
password admin
!
interface eth0
  ipv6 ospf6 cost 1
  ipv6 ospf6 hello-interval 10
  ipv6 ospf6 dead-interval 40
  ipv6 ospf6 retransmit-interval 5
  ipv6 ospf6 priority 1
  ipv6 ospf6 transmit-delay 1

interface eth1
  ipv6 ospf6 cost 1
  ipv6 ospf6 hello-interval 10
  ipv6 ospf6 dead-interval 40
  ipv6 ospf6 retransmit-interval 5
  ipv6 ospf6 priority 1
  ipv6 ospf6 transmit-delay 1

interface eth2
  ipv6 ospf6 cost 1
  ipv6 ospf6 hello-interval 10
  ipv6 ospf6 dead-interval 40
  ipv6 ospf6 retransmit-interval 5
  ipv6 ospf6 priority 1
  ipv6 ospf6 transmit-delay 1

interface eth3
  ipv6 ospf6 cost 1
  ipv6 ospf6 hello-interval 10
  ipv6 ospf6 dead-interval 40
  ipv6 ospf6 retransmit-interval 5
  ipv6 ospf6 priority 1
  ipv6 ospf6 transmit-delay 1
!
router ospf6
  area 0.0.0.1 range 2001:aaaa:cccc:2::/64
  area 0.0.0.1 range 2001:aaaa:cccc:5::/64
  area 0.0.0.1 range 2001:aaaa:cccc:6::/64
  area 0.0.0.1 range 2001:aaaa:dddd:1::/64
  interface eth0 area 0.0.0.1
  interface eth1 area 0.0.0.1
  interface eth2 area 0.0.0.1
  interface eth3 area 0.0.0.1

```

8.1) en *ospf6d.conf*, en la línea con el comando *router ospf6* se está habilitando el algoritmo de enrutamiento ospfv3, y en *zebra.conf* se habilita el reenvío de paquetes IPv6 con el comando *ipv6 forwarding*.

9) Para verificar que ospf configuró correctamente los routers, se hacen pruebas de conectividad (ping) entre distintos puntos en la red (en la siguiente captura se muestran algunas pruebas):


```

root@ivanovic-VirtualBox:/home/ivanovic/docker_quagga/ospf# docker exec -it ospf_r2_1 /bin/bash
root@f73ed09ed222:/go# ping6 2001:aaaa:cccc:1::10
PING 2001:aaaa:cccc:1::10 (2001:aaaa:cccc:1::10): 56 data bytes
64 bytes from 2001:aaaa:cccc:1::10: icmp_seq=0 ttl=63 time=0.428 ms
64 bytes from 2001:aaaa:cccc:1::10: icmp_seq=1 ttl=63 time=0.285 ms
64 bytes from 2001:aaaa:cccc:1::10: icmp_seq=2 ttl=63 time=0.288 ms
^Z
[1]+  Stopped                  ping6 2001:aaaa:cccc:1::10
root@f73ed09ed222:/go# exit
exit
There are stopped jobs.
root@f73ed09ed222:/go# exit
exit
root@ivanovic-VirtualBox:/home/ivanovic/docker_quagga/ospf# docker exec -it ospf_h4_1 /bin/bash
root@d881c486ab0b:/go# ping6 2001:aaaa:bbbb:6::2
PING 2001:aaaa:bbbb:6::2 (2001:aaaa:bbbb:6::2): 56 data bytes
64 bytes from 2001:aaaa:bbbb:6::2: icmp_seq=0 ttl=61 time=0.244 ms
64 bytes from 2001:aaaa:bbbb:6::2: icmp_seq=1 ttl=61 time=0.281 ms
^Z
[1]+  Stopped                  ping6 2001:aaaa:bbbb:6::2
root@d881c486ab0b:/go# exit
exit
There are stopped jobs.
root@d881c486ab0b:/go# exit
exit
root@ivanovic-VirtualBox:/home/ivanovic/docker_quagga/ospf# docker exec -it ospf_r5_1 /bin/bash
root@1fe553978b40:/go# ping6 2001:aaaa:bbbb:3::20
PING 2001:aaaa:bbbb:3::20 (2001:aaaa:bbbb:3::20): 56 data bytes
64 bytes from 2001:aaaa:bbbb:3::20: icmp_seq=0 ttl=63 time=0.404 ms
64 bytes from 2001:aaaa:bbbb:3::20: icmp_seq=1 ttl=63 time=0.127 ms
64 bytes from 2001:aaaa:bbbb:3::20: icmp_seq=2 ttl=63 time=0.130 ms
^Z
[1]+  Stopped                  ping6 2001:aaaa:bbbb:3::20
root@1fe553978b40:/go#

```

para ver que las tablas de ruteo muestren las rutas ospf, conectando con telnet a un router (r1 por ejemplo), se ejecuta el comando *show ipv6 ospf6 route*:

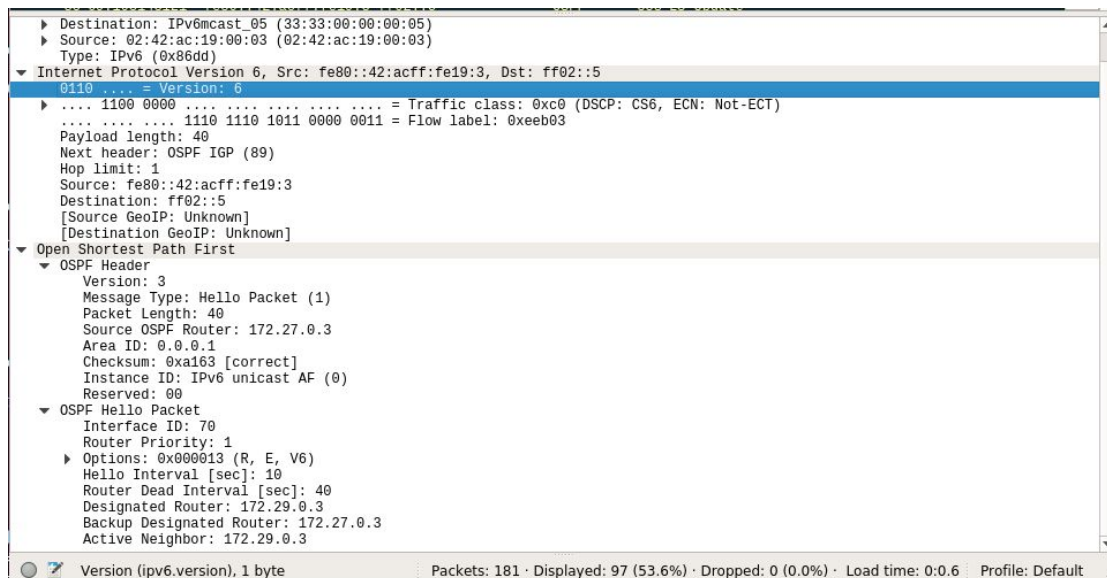
```

r1> show ipv6 ospf6 route
*N IA 2001:aaaa:bbbb:1::/64      fe80::42:acff:fe12:3      eth0 01:41:35
*N IA 2001:aaaa:bbbb:2::/64      ::                        eth0 01:41:40
*N IA 2001:aaaa:bbbb:3::/64      fe80::42:acff:fe13:2      eth1 01:41:35
*N IA 2001:aaaa:bbbb:4::/64      fe80::42:acff:fe12:3      eth0 01:41:35
                                fe80::42:acff:fe13:2      eth1
*N IA 2001:aaaa:bbbb:5::/64      ::                        eth1 01:41:35
*N IA 2001:aaaa:bbbb:6::/64      ::                        eth2 01:41:40
*N IA 2001:aaaa:cccc:1::/64      fe80::42:acff:fe15:2      eth3 01:38:50
*N IA 2001:aaaa:cccc:2::/64      fe80::42:acff:fe15:2      eth3 01:39:25
*N IA 2001:aaaa:cccc:3::/64      fe80::42:acff:fe15:2      eth3 01:38:50
*N IA 2001:aaaa:cccc:4::/64      fe80::42:acff:fe15:2      eth3 01:38:49
*N IA 2001:aaaa:cccc:5::/64      fe80::42:acff:fe15:2      eth3 01:39:30
*N IA 2001:aaaa:cccc:6::/64      fe80::42:acff:fe15:2      eth3 01:41:35
*N IA 2001:aaaa:dddd:1::/64      ::                        eth3 01:41:40

```

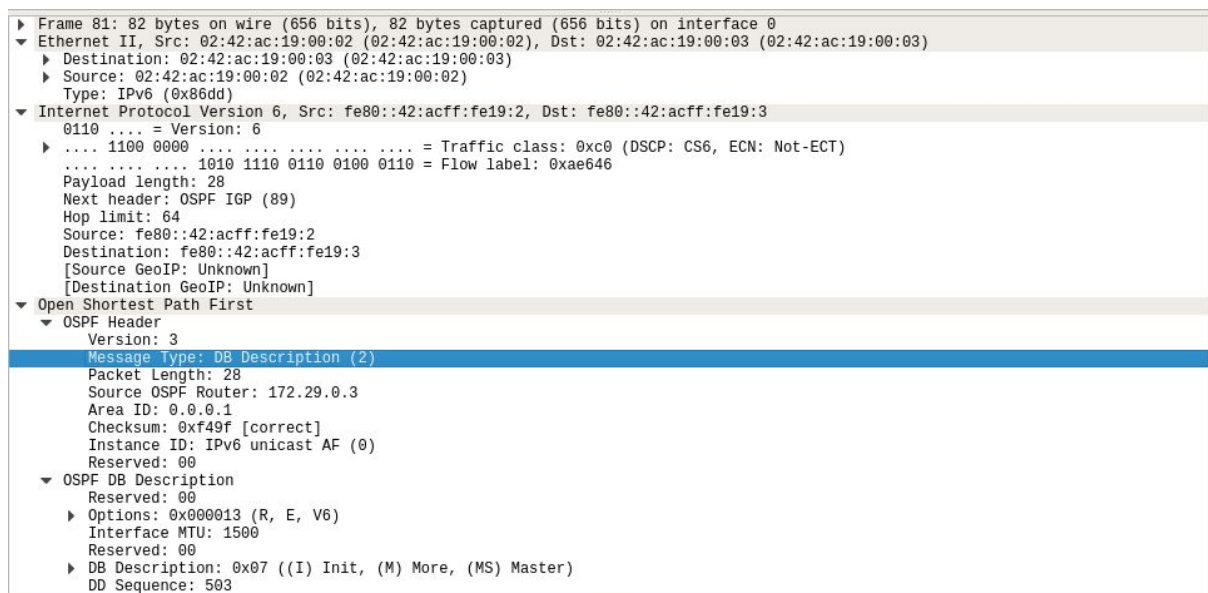
10) Con wireshark, se hicieron capturas desde la inicialización de los contenedores. De esta forma, se pudieron ver los 5 tipos de mensajes que utiliza ospf. A continuación, se muestran capturas de cada uno de ellos, y una breve explicación:

- Tipo 1: hello packet



Permite el descubrimiento dinámico de vecinos y mantiene relaciones vecinas. Envía paquetes a la dirección de multidifusión FF02::5. En esta configuración, envía cada 10 segundos.

- Tipo 2: Descriptor de base de datos (LSDB)



Se emplean en el intercambio de base de datos enlace-estado entre dos nodos, y permiten informar al otro nodo implicado en la sincronización acerca de los registros contenidos en la LSDB propia, mediante un resumen de estos.

- Tipo 3: Paquete de solicitud de estado de enlace (LSR)


```

▶ Frame 83: 130 bytes on wire (1040 bits), 130 bytes captured (1040 bits) on interface 0
▼ 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)
  ▶ Destination: 02:42:ac:19:00:03 (02:42:ac:19:00:03)
  ▶ Source: 02:42:ac:19:00:02 (02:42:ac:19:00:02)
  Type: IPv6 (0x86dd)
▼ Internet Protocol Version 6, Src: fe80::42:acff:fe19:2, Dst: fe80::42:acff:fe19:3
  0110 .... = Version: 6
  ▶ .... 1100 0000 .... = Traffic class: 0xc0 (DSCP: CS6, ECN: Not-ECT)
  ▶ .... 1010 1110 0110 0100 0110 = Flow label: 0xae646
  Payload length: 76
  Next header: OSPF IGP (89)
  Hop limit: 64
  Source: fe80::42:acff:fe19:2
  Destination: fe80::42:acff:fe19:3
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
▼ Open Shortest Path First
  ▼ OSPF Header
    Version: 3
    Message Type: LS Request (3)
    Packet Length: 76
    Source OSPF Router: 172.29.0.3
    Area ID: 0.0.0.1
    Checksum: 0x1f3c [correct]
    Instance ID: IPv6 unicast AF (0)
    Reserved: 00
    ▶ Link State Request
    ▶ Link State Request
    ▶ Link State Request
    ▶ Link State Request
    ▶ Link State Request
    ▶ Link State Request

```

Después de que el proceso de intercambio de paquetes LSDB, el enrutador puede encontrar que no tiene una base de datos actualizada. El paquete LSR se usa para solicitar piezas de la base de datos vecina que esté más actualizada.

- Tipo 4: Paquete de actualización de estado de enlace (LSU)

```

▶ Frame 85: 358 bytes on wire (2864 bits), 358 bytes captured (2864 bits) on interface 0
▼ Ethernet II, Src: 02:42:ac:19:00:03 (02:42:ac:19:00:03), Dst: IPv6mcast_05 (33:33:00:00:00:05)
  ▶ Destination: IPv6mcast_05 (33:33:00:00:00:05)
  ▶ Source: 02:42:ac:19:00:03 (02:42:ac:19:00:03)
  Type: IPv6 (0x86dd)
▼ Internet Protocol Version 6, Src: fe80::42:acff:fe19:3, Dst: ff02::5
  0110 .... = Version: 6
  ▶ .... 1100 0000 .... = Traffic class: 0xc0 (DSCP: CS6, ECN: Not-ECT)
  ▶ .... 1110 1110 1011 0000 0011 = Flow label: 0xeeb03
  Payload length: 304
  Next header: OSPF IGP (89)
  Hop limit: 1
  Source: fe80::42:acff:fe19:3
  Destination: ff02::5
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
▼ Open Shortest Path First
  ▼ OSPF Header
    Version: 3
    Message Type: LS Update (4)
    Packet Length: 304
    Source OSPF Router: 172.27.0.3
    Area ID: 0.0.0.1
    Checksum: 0xe5be [correct]
    Instance ID: IPv6 unicast AF (0)
    Reserved: 00
    ▶ LS Update Packet

```

Se utiliza para responder a los LSR y anunciar la nueva información. Los LSU contienen siete tipos de LSA.

- Tipo 5: paquete de acuse de recibo de estado de enlace (LSAck)

```

▶ Frame 100: 290 bytes on wire (2320 bits), 290 bytes captured (2320 bits) on interface 0
▼ Ethernet II, Src: 02:42:ac:19:00:03 (02:42:ac:19:00:03), Dst: IPv6mcast_05 (33:33:00:00:00:05)
  ▶ Destination: IPv6mcast_05 (33:33:00:00:00:05)
  ▶ Source: 02:42:ac:19:00:03 (02:42:ac:19:00:03)
  Type: IPv6 (0x86dd)
▼ Internet Protocol Version 6, Src: fe80::42:acff:fe19:3, Dst: ff02::5
  0110 .... = Version: 6
  ▶ .... 1100 0000 .... = Traffic class: 0xc0 (DSCP: CS6, ECN: Not-ECT)
  .... 1110 1110 1011 0000 0011 = Flow label: 0xeeb03
  Payload length: 236
  Next header: OSPF IGP (89)
  Hop limit: 1
  Source: fe80::42:acff:fe19:3
  Destination: ff02::5
  [Source GeoIP: Unknown]
  [Destination GeoIP: Unknown]
▼ Open Shortest Path First
  ▼ OSPF Header
    Version: 3
    Message Type: LS Acknowledge (5)
    Packet Length: 236
    Source OSPF Router: 172.27.0.3
    Area ID: 0.0.0.1
    Checksum: 0xe0f4 [correct]
    Instance ID: IPv6 unicast AF (0)
    Reserved: 00
  ▼ LSA-type 8 (Link-LSA), len 56
    .000 0000 0011 1100 = LS Age (seconds): 60
    0... .... = Do Not Age: False
    LS Type: Link-LSA (0x0008)
    Link State ID: 0.0.0.62
    Advertising Router: 172.29.0.3
    Sequence Number: 0x80000001

```

Cuando se recibe una LSU, el router envía un LSAck para confirmar la recepción de la LSU. El campo de datos del LSAck está vacío.

11) Eliminando la red 2001:aaaa:cccc:2::, que une directamente r5 y r6, para ir a la red 2001:aaaa:cccc:1:: se pasó de tener el próximo salto siguiente

```

root@8a136025c718:/go# ip -6 route
2001:aaaa:bbbb:1::/64 via fe80::42:acff:fe15:3 dev eth3 proto zebra metric 3
2001:aaaa:bbbb:2::/64 via fe80::42:acff:fe15:3 dev eth3 proto zebra metric 2
2001:aaaa:bbbb:3::/64 via fe80::42:acff:fe15:3 dev eth3 proto zebra metric 3
2001:aaaa:bbbb:4::/64 via fe80::42:acff:fe15:3 dev eth3 proto zebra metric 3
2001:aaaa:bbbb:5::/64 via fe80::42:acff:fe15:3 dev eth3 proto zebra metric 2
2001:aaaa:bbbb:6::/64 via fe80::42:acff:fe15:3 dev eth3 proto zebra metric 2
2001:aaaa:cccc:1::/64 via fe80::42:acff:fe19:2 dev eth1 proto zebra metric 2
2001:aaaa:cccc:2::/64 dev eth1 proto kernel metric 256
2001:aaaa:cccc:3::/64 via fe80::42:acff:fe1a:2 dev eth0 proto zebra metric 2
2001:aaaa:cccc:4::/64 via fe80::42:acff:fe19:2 dev eth1 proto zebra metric 2
2001:aaaa:cccc:5::/64 dev eth0 proto kernel metric 256
2001:aaaa:cccc:6::/64 dev eth2 proto kernel metric 256
2001:aaaa:dddd:1::/64 dev eth3 proto kernel metric 256
fe80::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth1 proto kernel metric 256
fe80::/64 dev eth2 proto kernel metric 256
fe80::/64 dev eth3 proto kernel metric 256

```

a tener este otro;

```

ivanovic@ivanovic-VirtualBox:~/docker_quagga/ospf$ sudo su
[sudo] password for ivanovic:
root@ivanovic-VirtualBox:/home/ivanovic/docker_quagga/ospf# docker exec -it ospf_r5_1 /bin/bash
root@a37bbb723094:/go# ip -6 route
2001:aaaa:bbbb:1::/64 via fe80::42:acff:fe15:3 dev eth1 proto zebra metric 6
2001:aaaa:bbbb:2::/64 via fe80::42:acff:fe15:3 dev eth1 proto zebra metric 5
2001:aaaa:bbbb:3::/64 via fe80::42:acff:fe15:3 dev eth1 proto zebra metric 6
2001:aaaa:bbbb:4::/64 via fe80::42:acff:fe15:3 dev eth1 proto zebra metric 6
2001:aaaa:bbbb:5::/64 via fe80::42:acff:fe15:3 dev eth1 proto zebra metric 5
2001:aaaa:bbbb:6::/64 via fe80::42:acff:fe15:3 dev eth1 proto zebra metric 5
2001:aaaa:cccc:1::/64 via fe80::42:acff:fe1a:2 dev eth0 proto zebra metric 3
2001:aaaa:cccc:2::/64 via fe80::42:acff:fe1a:2 dev eth0 proto zebra metric 3
2001:aaaa:cccc:3::/64 via fe80::42:acff:fe1a:2 dev eth0 proto zebra metric 2
2001:aaaa:cccc:4::/64 via fe80::42:acff:fe1a:2 dev eth0 proto zebra metric 2
2001:aaaa:cccc:5::/64 dev eth0 proto kernel metric 256
2001:aaaa:cccc:6::/64 dev eth2 proto kernel metric 256
2001:aaaa:dddd:1::/64 dev eth1 proto kernel metric 256
fe80::/64 dev eth0 proto kernel metric 256
fe80::/64 dev eth1 proto kernel metric 256
fe80::/64 dev eth2 proto kernel metric 256
default via 2001:aaaa:cccc:5::1 dev eth0 metric 1024
root@a37bbb723094:/go#

```

Ahora los paquetes dirigidos a h3, pasaran antes por r2 en lugar de r6.

ANEXO: Se hizo una prueba con traceroute desde el router r2 hasta el host h4:

```

traceroute to 2001:aaaa:cccc:3::20 (2001:aaaa:cccc:3::20), 30 hops max, 80 byte
packets
 1 ospf_r1_1.ospf_red2 (2001:aaaa:bbbb:2::2) 0.135 ms 0.010 ms 0.009 ms
 2 2001:aaaa:dddd:1::2 (2001:aaaa:dddd:1::2) 0.034 ms 0.015 ms 0.015 ms
 3 2001:aaaa:cccc:5::3 (2001:aaaa:cccc:5::3) 0.066 ms 0.017 ms 0.016 ms
 4 2001:aaaa:cccc:3::20 (2001:aaaa:cccc:3::20) 0.093 ms 0.019 ms 0.016 ms
root@f73ed09ed222:/go#

```

links:

<http://www.itesa.edu.mx/netacad/switching/course/module8/8.1.2.2/8.1.2.2.html>

<https://sites.google.com/site/amitsciscozone/home/important-tips/ospf/ospf-packet-types>

https://www.youtube.com/watch?v=9Rta8qqrkpc&list=PLn5IkU1ZhgiZI4EH7AFkqs-pqF6ZUz_iS&index=15

<http://www.ticarte.com/contenido/quagga-enrutamiento-ospf>

https://www.watchguard.com/help/docs/fireware/12/es-419/Content/es-419/dynamicrouting/ospfv3_commands_c.html