Redes y Comunicaciones

Ulises Jeremias Cornejo Fandos,[1] Federico Ramón Gasquez,[2] and Lihuel Pablo Amoroso[3]

[1] *13566/7, Licenciatura en Informatica, Facultad de Informatica, UNLP*
[2] *13598/6, Licenciatura en Informatica, Facultad de Informatica, UNLP*
[3] *13497/2, Analista Programador Universitario, Facultad de Informatica, UNLP*

compiled: November 6, 2018

1. Ejercicio 1

Utilizando topología topologia-IP.imn y dado el bloque IPv6: 2001:db8:1234::/48.

a) Arme el plan de direccionamiento IPv6 teniendo en cuenta las siguientes restricciones:

- La red A tiene 70 hosts y se espera un crecimiento máximo de 20 hosts.
- La red X tiene 150 hosts.
- La red B cuenta con 20 hosts
- La red Y tiene 35 hosts y se espera un crecimiento máximo de 30 hosts.
- Los bloques IP asignados en los enlaces entre routers podrán ajustarse a desperdiciar pocas direcciones.
- Es importante utilizar VLSM?

b) Asigne direcciones IP en los equipos de la topología según el plan anterior.

c) Configure las tablas de rutas teniendo en cuenta las siguientes restricciones:

d) n1 deberá optar por el enlace verde solamente para rutear el tráfico dirigido a la Red X.

e) Utilizando la herramienta ping6(8), verifique conectividad entre los hosts pertenecientes a las diferentes redes de usuarios.

Se adjunta el archivo *resources/topologia.imn* en la cual se encuentra la topologia de la red diseñada cumpliendo todos los requisitos. En la misma no solo se encuentran las asignaciones de IPv6, sino también las IPv4 correspondientes al último enunciado.

Para armar el plan de direccionamiento IPv6 de esta topologia no es necesario utilizar VLSM dado que la cantidad de host a asignar por red nunca llegan a ser tantas como para que no alcanse la cantidad de IPs asignables.

2. Ejercicio 2

TTL (Adjunte capturas de tráfico para cada uno de los incisos).

a) *Utilizando el comando **traceroute6/tracepath6(8)**, realice una traza entre el host n8 y n10, tanto utilizando UDP como ICMP. ¿Qué diferencias tiene cada método y en qué casos utilizaría cada uno?.*

El comando **traceroute6** envia en ambos casos, utilizando tanto UDP como ICMP, un paquete con el valor del campo TTL *(time to live)* de la cabecera IP igual a 1, esperando una respuesta ICMP de la forma *Time Exceeded* ó *Time to Live Expired* del primer router que recibe el mensaje, ya que en el primer salto el TTL se habrá decrementado a 0. Luego se incrementa el TTL en 1, esperando recibir una respuesta del segundo router involucrado, y se repite el proceso para construir el camino hasta recibir una respuesta del destino al que se quería conocer inicialmente.

La diferencia radica en que, bajo el protocolo ICMP, se envian paquetes del tipo *echo request* esperando que, una vez que se llegue al destino, este último envie un *echo reply*. El problema es que puede que algunos routers

filtren los mensajes de tipo echo request, imposibilitando encontrar a través de este protocolo la ruta completa; en este caso conviene realizar el traceroute a través de UDP. Este último envía mensajes de tipo UDP a un puerto destino válido, y una vez que llega al router destino, este le enviara un mensaje indicando que el puerto destino es inalcanzable.

```
root@n8:/tmp/pycore.57686/n8.conf# traceroute6 -I 2001:db8:1234:3::3
traceroute to 2001:db8:1234:3::3 (2001:db8:1234:3::3), 30 hops max, 80 byte packets
 1  2001:db8:1234:1::1 (2001:db8:1234:1::1)  0.069 ms  0.012 ms  0.009 ms
 2  2001:db8:1234:9::1 (2001:db8:1234:9::1)  0.075 ms  0.015 ms  0.012 ms
 3  2001:db8:1234:7::1 (2001:db8:1234:7::1)  0.185 ms  0.026 ms  0.020 ms
 4  2001:db8:1234:5::1 (2001:db8:1234:5::1)  0.090 ms  0.023 ms  0.023 ms
 5  2001:db8:1234:3::3 (2001:db8:1234:3::3)  0.038 ms  0.023 ms  0.022 ms
root@n8:/tmp/pycore.57686/n8.conf#
```

Fig. 1. Traza entre el comando n8 y n10 utilizando traceroute6 con ICMP.

```
root@n8:/tmp/pycore.57686/n8.conf# traceroute6 -U 2001:db8:1234:3::3
traceroute to 2001:db8:1234:3::3 (2001:db8:1234:3::3), 30 hops max, 80 byte packets
 1  2001:db8:1234:1::1 (2001:db8:1234:1::1)  0.056 ms  0.007 ms  0.006 ms
 2  2001:db8:1234:9::1 (2001:db8:1234:9::1)  0.018 ms  0.009 ms  0.008 ms
 3  2001:db8:1234:7::1 (2001:db8:1234:7::1)  0.029 ms  0.012 ms  0.012 ms
 4  2001:db8:1234:5::1 (2001:db8:1234:5::1)  0.025 ms  0.014 ms  0.014 ms
 5  2001:db8:1234:3::3 (2001:db8:1234:3::3)  0.023 ms  0.016 ms  0.014 ms
root@n8:/tmp/pycore.57686/n8.conf#
```

Fig. 2. Traza entre el comando n8 y n10 utilizando traceroute6 con UDP.

Se adjunta el tráfico captura en los archivos *resources/traceroute6icmp.pcapng* y *resources/traceroute6ucp.pcapng*.

b) *Realice un ping entre n8 y n5 y determine el valor inicial del campo TTL capturando tráfico en la interfaz eth0 del host n8.*

Para este inciso, se ejecuta en la terminal correspondiente al host n8 un ping entre este host y n7 utilizando el comando *ping6*.

Capturando el tráfico con el wireshark se busca determinar el valor del campo TTL en el header del paquete IPv6. Tras no poder encontrarlo se inspecciona el formato de una cabecera IPv6 y concluyendo que dicha información se puede obtener en el header Hop Limit. Leer RFC 2460.

Se puede observar entonces en la figura 3, que el valor inicial del campo TTL, *Hop Limit en IPv6*, es 64 en la interfaz eth0 del host n8.

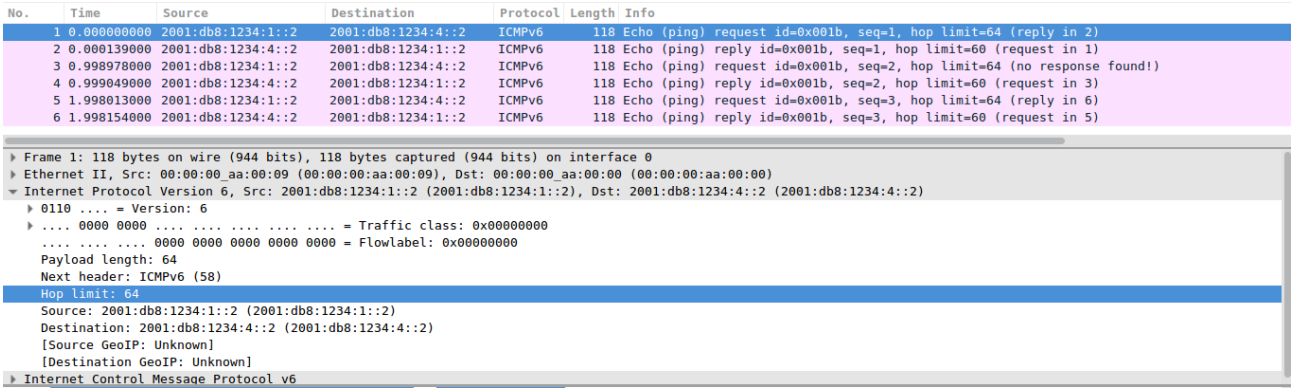

Fig. 3. Captura del tráfico del ping entre n8 y n7 en el que se observa el valor inicial del campo Hop Limit.

c) *A través de la capturas de tráfico, determine en qué momento el router decrementa el valor del TTL.*

Dada la ejecución del comando presentado en la figura 4 se pueden apreciar las capturas de paquetes de cada uno de los routers en las figuras 5, 6, 7 y 8.

```
root@n8:/tmp/pycore.44887/n8.conf# ping6 -c 3 2001:db8:1234:4::2
PING 2001:db8:1234:4::2(2001:db8:1234:4::2) 56 data bytes
64 bytes from 2001:db8:1234:4::2: icmp_seq=1 ttl=60 time=0.120 ms
64 bytes from 2001:db8:1234:4::2: icmp_seq=2 ttl=60 time=0.111 ms
64 bytes from 2001:db8:1234:4::2: icmp_seq=3 ttl=60 time=0.117 ms

--- 2001:db8:1234:4::2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.111/0.116/0.120/0.003 ms
```

Fig. 4. Salida de la ejecución del comando ping realizado del nodo n8 al n7.

Observando las capturas de la interfaz de salida de cada uno de los routers a través de los cuales viaja el ping, se ve como el TTL, *Hop Limit* en IPv6, se decrementa con respecto a la captura del mismo router en la interfaz de entrada del mensaje, es decir, que el router lo decrementa previo al forwarding.

```
root@n1:/tmp/pycore.44887/n1.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:59:14.122882 IP6 (hlim 64, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:1::2 > 2001:db8:1234:4::2: [icmp6 sum ok] ICMP6, echo request, seq 1
20:59:14.122969 IP6 (hlim 60, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:4::2 > 2001:db8:1234:1::2: [icmp6 sum ok] ICMP6, echo reply, seq 1
```

Fig. 5. Captura realizada desde el router n1 del tráfico entre n8 y n7.

```
root@n11:/tmp/pycore.44887/n11.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:59:14.122899 IP6 (hlim 63, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:1::2 > 2001:db8:1234:4::2: [icmp6 sum ok] ICMP6, echo request, seq 1
20:59:14.122962 IP6 (hlim 61, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:4::2 > 2001:db8:1234:1::2: [icmp6 sum ok] ICMP6, echo reply, seq 1
```

Fig. 6. Captura realizada desde el router n11 del tráfico entre n8 y n7.

```
root@n6:/tmp/pycore.44887/n6.conf# tcpdump -v -i eth1
tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
20:59:14.122917 IP6 (hlim 61, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:1::2 > 2001:db8:1234:4::2: [icmp6 sum ok] ICMP6, echo request, seq 1
20:59:14.122952 IP6 (hlim 63, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:4::2 > 2001:db8:1234:1::2: [icmp6 sum ok] ICMP6, echo reply, seq 1
```

Fig. 7. Captura realizada desde el router n6 del tráfico entre n8 y n7.

```
root@n3:/tmp/pycore.44887/n3.conf# tcpdump -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:59:14.122922 IP6 (hlim 61, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:1::2 > 2001:db8:1234:4::2: [icmp6 sum ok] ICMP6, echo request, seq 1
20:59:14.122950 IP6 (hlim 63, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:4::2 > 2001:db8:1234:1::2: [icmp6 sum ok] ICMP6, echo reply, seq 1
```

Fig. 8. Captura realizada desde el router n3 del tráfico entre n8 y n7.

d) *Utilizando la herramienta para enviar mensajes ICMP con la opción -t desde n8 envíe un datagrama a n7 con TTL=1. ¿Qué mensaje recibe? ¿Por qué?*

Para enviar mensajes ICMP se utiliza el comando ping(6) ejecutando, en la terminal del nodo n8, el siguiente comando,

```
$ ping6 -t 2001:db8:1234:4::2
```

, siendo 2001:db8:1234:4::2 la IPv6 del nodo n7.

A la par que se envía el datagrama, se captura el tráfico utilizando wireshark como se observa en la figura (9). El mensaje que se recibe tanto en la salida del comando ejecutado (figura 10) como en los paquetes capturados por el wireshark es *Time exceeded: Hop Limit*.

Dado que el valor inicial del TTL es 1, el primer salto cuando va hacia el primer router de la red, decrementa el TTL, llevando su valor a 0 y en consecuencia, descartando el paquete, posteriormente le envía un mensaje ICMP al host indicándole que se excedió el tiempo del TTL, *Time exceeded: Hop Limit*.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	2001:db8:1234:1::2	2001:db8:1234:4::2	ICMPv6	118	Echo (ping) request id=0x0029, seq=1, hop limit=1 (no response found!)
2	0.000049000	2001:db8:1234:1::1	2001:db8:1234:1::2	ICMPv6	166	Time Exceeded (hop limit exceeded in transit)
3	1.007283000	2001:db8:1234:1::2	2001:db8:1234:4::2	ICMPv6	118	Echo (ping) request id=0x0029, seq=2, hop limit=1 (no response found!)
4	1.007388000	2001:db8:1234:1::1	2001:db8:1234:1::2	ICMPv6	166	Time Exceeded (hop limit exceeded in transit)
5	2.015064000	2001:db8:1234:1::2	2001:db8:1234:4::2	ICMPv6	118	Echo (ping) request id=0x0029, seq=3, hop limit=1 (no response found!)
6	2.015110000	2001:db8:1234:1::1	2001:db8:1234:1::2	ICMPv6	166	Time Exceeded (hop limit exceeded in transit)
7	3.023623000	2001:db8:1234:1::2	2001:db8:1234:4::2	ICMPv6	118	Echo (ping) request id=0x0029, seq=4, hop limit=1 (no response found!)
8	3.023716000	2001:db8:1234:1::1	2001:db8:1234:1::2	ICMPv6	166	Time Exceeded (hop limit exceeded in transit)
9	4.031071000	2001:db8:1234:1::2	2001:db8:1234:4::2	ICMPv6	118	Echo (ping) request id=0x0029, seq=5, hop limit=1 (no response found!)
10	4.031120000	2001:db8:1234:1::1	2001:db8:1234:1::2	ICMPv6	166	Time Exceeded (hop limit exceeded in transit)

Fig. 9. Captura del tráfico del ping entre n8 y n7.

```
root@n8:/tmp/pycore.36269/n8.conf# ping6 -c 5 -t 1 2001:db8:1234:4::2
PING 2001:db8:1234:4::2(2001:db8:1234:4::2) 56 data bytes
From 2001:db8:1234:1::1 icmp_seq=1 Time exceeded: Hop limit
From 2001:db8:1234:1::1 icmp_seq=2 Time exceeded: Hop limit
From 2001:db8:1234:1::1 icmp_seq=3 Time exceeded: Hop limit
From 2001:db8:1234:1::1 icmp_seq=4 Time exceeded: Hop limit
From 2001:db8:1234:1::1 icmp_seq=5 Time exceeded: Hop limit

--- 2001:db8:1234:4::2 ping statistics ---
5 packets transmitted, 0 received, +5 errors, 100% packet loss, time 4031ms
```

Fig. 10. Salida del comando ping.

Se adjunta la captura del tráfico realizada utilizando wireshark en el archivo *resources/ttl1.pcapng*.

3. Ejercicio 3

Dual Stack

a) Configure en la parte “X” y en “B” la red en IPv4 e IPv6 en modo Dual Stack.

Para configurar las partes ”X” y ”B” de la red, asignamos IPv4 a los host de cada red y a las interfaces del router. Para esto se asignan, e.g. IPv4 privadas de la clase A, aplicando VLSM sobre cada una de las partes.

Dado el bloque IPv4, 10.0.0.0 (con mascara de subred 8 por ser la mascara de clase A), se necesitan IPs para 150 hosts en la Red X y 20 hosts para la Red B, quedando las siguientes IPv4 para cada una de las redes respectivamente:

1. Red X tiene asignada el bloque IPv4 10.0.0.0/24 con 254 hosts disponibles.
2. Red B tiene asignada el bloque IPv4 10.0.1.0/27 con 30 hosts disponibles.

Luego, quedan las siguientes IP de host,

1. Red X, router n3, eth2, 10.0.0.1/24
2. Red X, host n10, eth0, 10.0.0.253/24
3. Red X, host n12, eth0, 10.0.0.254/24
4. Red B, router n3, eth3, 10.0.1.1/27
5. Red B, host n15, eth0, 10.0.1.29/27
6. Red B, host dhcp-server, eth0, 10.0.1.30/27

b) Genere tráfico IPv6 e IPv4 y compare.

Analizando las capturas de las figuras 11 y 12 podemos observar la presencia del header offset en el tráfico de IPv4, que no se encuentra en las capturas de IPv6. A su vez, aparece el flag DF que en las capturas de IPv6 no está. El mismo hace pensar sobre la presencia de la Fragmentación en IPv4. El flag **more fragments** indica

que el datagrama es en realidad un fragmento de un datagrama más grande y lo suceden más fragmentos, y el campo **offset** indica la posición (en bytes) del fragmento de datagrama con respecto al datagrama que compone. En este caso, ambos están en 0, ya que no hubo necesidad de fragmentación, pero en caso de que se envie un datagrama más grande (utilizando el comando *ping* con el flag *s* para definir un tamaño mayor de datagrama), habrá fragmentacion y ambos campos tendrían el valor correspondiente.

La **fragmentación IP** entonces es un mecanismo que permite separar *(o fragmentar)* un paquete IP entre varios bloques de datos, si su tamaño sobrepasa la unidad máxima de transferencia (Maximum Transfer Unit - MTU) del canal.

Aunque el objetivo es una implementación para capas más altas (por ejemplo TCP/UDP) este no está conseguido en dos puntos:

- La fragmentación puede tener una gran influencia negativa en la actuación y en el flujo de datos.
- Si se pierde un fragmento, hay que retransmitir todos los fragmentos del paquete original. Sin embargo IP no tiene mecanismos de seguridad o de timeout y es dependiente de las funciones de seguridad de las capas más altas como TCP.

Por las razones mencionadas se intenta de evitar la fragmentación siempre que sea posible.

En el caso de IPv6, ya no permite a los routers fragmentar los paquetes. El emisor siempre está informado con un mensaje ICMP cuando una fragmentación será necesaria. Así el emisor puede bajar su tamaño de paquete para esta conexión y la fragmentación ya no es necesaria. En caso de requerirse una fragmentación; el host, es quien debe hacerla.

```
22:23:34.436066 IP (tos 0x0, ttl 64, id 19527, offset 0, flags [DF], proto ICMP (1), length 84)
    10.0.0.253 > 10.0.1.30: ICMP echo request, id 30, seq 1, length 64
22:23:34.436110 IP (tos 0x0, ttl 63, id 60638, offset 0, flags [none], proto ICMP (1), length 84)
    10.0.1.30 > 10.0.0.253: ICMP echo reply, id 30, seq 1, length 64
```

Fig. 11. Ejemplo de tráfico en IPv4 capturado por tcpdump.

```
22:26:11.583676 IP6 (hlim 64, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:3::3 > 2001:db8:1234:2::2: [icmp6 sum ok] ICMP6, echo request, seq 1
22:26:11.583739 IP6 (hlim 63, next-header ICMPv6 (58) payload length: 64) 2001:db8:1234:2::2 > 2001:db8:1234:3::3: [icmp6 sum ok] ICMP6, echo reply, seq 1
```

Fig. 12. Ejemplo de tráfico en IPv6 capturado por tcpdump.

c) Investigue mecanismos para llevar el tráfico IPv4 de la red “X” a la red “B” pasando por redes. Vea cómo puede implementarse en la topología.

Investigando otros mecanismos nos encontramos con el siguiente post: IPv6: Dual stack where you can; tunnel where you must.

En el mismo se describe como IPv6 se entregó con técnicas de migración para cubrir todos los casos de actualización de IPv4 concebibles, pero muchos fueron rechazados en última instancia por la comunidad tecnológica, y hoy nos queda un pequeño conjunto de enfoques prácticos.

Una técnica, llamada pila dual, implica ejecutar IPv4 e IPv6 al mismo tiempo. Los nodos finales y los enrutadores / conmutadores ejecutan ambos protocolos, y si la comunicación IPv6 es posible, ese es el protocolo preferido.

Otro enfoque es usar túneles para llevar un protocolo dentro de otro. Estos túneles toman paquetes de IPv6 y los encapsulan en paquetes de IPv4 para enviarlos a través de partes de la red que aún no se han actualizado a IPv6. Se pueden crear túneles donde haya islas IPv6 separadas por un océano IPv4, que será la norma durante las primeras etapas de la transición a IPv6. Más adelante habrá islas IPv4 que deberán ser puenteadas a través de un océano IPv6.

Otras técnicas, como la traducción de direcciones de red y la traducción de protocolos (NAT-PT) simplemente traducen los paquetes IPv6 en paquetes IPv4. Estas técnicas de traducción son más complicadas que IPv4 NAT porque los protocolos tienen diferentes formatos de encabezado. Las técnicas de traducción estaban destinadas a ser utilizadas como último recurso.

El uso de técnicas de dual stack y tunneling es preferible al uso de NAT-PT.