Práctica/Laboratorio de IPv4 (1)

Amoroso, Lihuel Pablo 13497/2; Gasquez, Federico Ramón 13598/6

Grupo A

1. Utilizando la topología topologia-IP.imn y dado el bloque IPv4 asignado resolver:

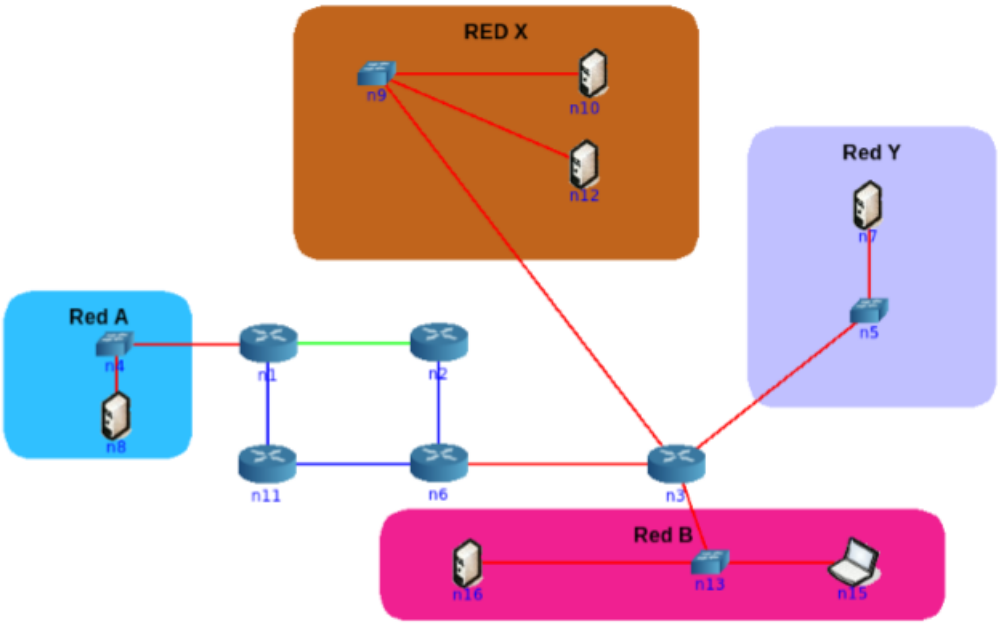

Figura 1: Esquema a subnetear

1.1. Arme el plan de direccionamiento IPv4 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?

  Si, ya que de esa manera se desperdician menos direcciones.

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

Si bien se adjunta la topología de forma que se pueda efectivamente verificar que las direcciones han sido correctamente asignadas, se procederá a detallar el procedimiento utilizado.

Primero ha de determinarse la red con más hosts. Esto se realiza para poder determinar cuántos bits se utilizarán como máximo. En este caso, la red con más hosts es la red X con 150.

De ahí, los pasos a seguir para asignar direcciones IP a los equipos en la topología se pueden reducir en los siguientes:

- Determinar la red con más cantidad de hosts.
- Usar una de las redes sobrantes para subnetear.
- Determinar cuáles son las redes sobrantes para el próximo subnet(si hubiera).

Aplicando el concepto al problema, tenemos la dirección 51.37.192.0/19. La red con más hosts, como ya se concluyó antes, es la red X con 150 hosts. Para representar tal cantidad, son necesarios 8 bits(con 7 bits, hay 24 hosts que se me quedan afuera) con un desperdicio de 104 hosts. Como voy a usar 8 bits, voy a correr la máscara a 24 en lugar de dejarla en 19. Concluyendo, la dirección que se asignará a la red X será la 51.37.192.0/24. Sobrarán, de esta forma, 32 direcciones de red y la siguiente a la asignada a la red X(51.37.193.0/24) será la próxima a subnetear.

Habiéndole asignado una dirección a la red X, la red con más hosts ahora es la red A con 90 hosts. Para representar tal cantidad, son necesarios 7 bits(con 6 bits, hay 28 hosts que se me quedan afuera) con un desperdicio de 36 hosts. Como voy a usar 7 bits, voy a correr la máscara a 25 en lugar de dejarla en 24. Concluyendo, la dirección que se asignará a la red A será la 51.37.193.0/25. Sobrará, de este modo, 1 dirección de red(51.37.193.128/25) que será la próxima a subnetear.

Habiéndole asignado una dirección a la red X y A, la red con más hosts es ahora la red Y con 65 hosts. Para representar tal cantidad, son necesarios 7 bits(con 6 bits, hay 3 hosts que se me quedan afuera) con un desperdicio de 61 hosts. Como voy a usar 7 bits, la máscara queda intacta(en /25). Concluyendo, la dirección que se asignará a la red Y será la 51.37.193.128/25. En este caso, no sobran más direcciones, por lo que será necesario tomar la siguiente a la siguiente de la red A(51.37.194.0/24) y será ésta la próxima a subnetear.

Habiéndole asignado una dirección a la red X, A y Y, la red con más hosts ahora(y la última que resta) es la red B con 20 hosts. Para representar tal cantidad, son necesarios 5 bits(con 4 bits, hay 6 hosts que se me quedan afuera) con un desperdicio de 10 hosts. Como voy a usar 5 bits, voy a correr la máscara a 27 en lugar de dejarla en 24. Concluyendo, la dirección que se asignará a la red B será la 51.37.194.0/27. Sobrarán, así, 7 direcciones de red y la siguiente a la asignada a la red B(51.37.194.32/27)será la próxima a subnetear.

Habiéndose ya asignado una dirección a las redes principales(red A, B, X e Y), es que ahora toca asignar direcciones a los enlaces punto a punto. Para asignar direcciones IP punto a punto débese tener en cuenta que solo son necesarios 4 hosts: el host reservado, el extremo A, el extremo B y el host de broadcast. Por esto es que serán necesarios 2 bits(con 1 bit, hay dos hosts que se me quedan afuera) para tal representación. Así mismo, existen 5 enlaces punto a punto: n1-n2, n1-n11, n2-n6, n6-n11 y n6-n3, por lo que se necesitan 3 bits(con 2 bits, hay 1 enlace punto a punto que no puedo representar) con un desperdicio de 3 redes. Teniendo la dirección 51.37.194.32/27, voy a correr la máscara de 27 a 30, indicando que solo utilizaré 4 hosts. De 27 a 30 hay 3, es decir, hay 3 bits que podré utilizar como redes a enlaces punto a punto, esto es, podré asignar 8 redes. Son necesarias 5 redes para los enlaces punto a punto, por lo que puedo continuar(podría haber pasado que no hubieran suficientes bits para representar las redes punto a punto. En tal caso hubiera sido necesario usar la dirección siguiente a la siguiente a la siguiente de la red A, es decir, la 51.37.195.0/24). Concluyendo, asignaré la red 51.37.194.32/30 al enlace n1-n2, 51.37.194.36/30 al enlace n1-n11, 51.37.194.40/30 al enlace n2-n6, 51.37.194.44/30 al enlace n6-n11 y 51.37.194.48/30 al enlace n6-n3.

Pasando en limpio(y agregando las direcciones asignadas a los hosts):

- Red X: 51.37.192.0/24.
  - n3(router-eth2): 51.37.192.1/24.
  - n10(host-eth0): 51.37.192.2/24.
  - n12(host-eth0): 51.37.192.3/24.
- Red A: 51.37.193.0/25.
  - n1(router-eth0): 51.37.193.1/25.
  - n8(host-eth0): 51.37.193.2/25.
- Red Y: 51.37.193.128/25.
  - n3(router-eth1): 51.37.193.129/25.
  - n7(host-eth0): 51.37.193.130/25.
- Red B: 51.37.194.0/27.
  - n3(router-eth3): 51.37.194.1/27.
  - n15(host-eth0): 51.37.194.2/27.
  - dhcp-server(host-eth0): 51.37.194.3/27.
- Enlace punto-a-punto n1-n2: 51.37.194.32/30.
  - n1(router-eth1): 51.37.194.33/30.
  - n2(router-eth0): 51.37.194.34/30.
- Enlace punto-a-punto n1-n11: 51.37.194.36/30.
  - n1(router-eth2): 51.37.194.37/30.
  - n11(router-eth0): 51.37.194.38/30.

- Enlace punto-a-punto n2-n6: 51.37.194.40/30.
  - n2(router-eth1): 51.37.194.41/30.
  - n6(router-eth0): 51.37.194.42/30.
- Enlace punto-a-punto n6-n11: 51.37.194.44/30.
  - n6(router-eth2): 51.37.194.45/30.
  - n11(router-eth1): 51.37.194.46/30.
- Enlace punto-a-punto n6-n3: 51.37.194.48/30.
  - n6(router-eth1): 51.37.194.49/30.
  - n3(router-eth0): 51.37.194.50/30.

1.3. Configure las tablas de rutas teniendo en cuenta que n1 deberá optar por el enlace verde solamente para rutear el tráfico dirigido a la Red X.

- n8

Destination	Gateway	GenMask	Iface
0.0.0.0	51.37.193.1	0.0.0.0	eth0

- n1

Destination	Gateway	GenMask	Iface
51.37.192.0	51.37.194.34	255.255.255.0	eth1
51.37.193.0	0.0.0.0	255.255.255.128	eth0
51.37.193.128	51.37.194.38	255.255.255.128	eth0
51.37.194.0	51.37.194.38	255.255.255.224	eth2
51.37.194.32	0.0.0.0	255.255.255.252	eth1
51.37.194.36	0.0.0.0	255.255.255.252	eth2
51.37.194.40	51.37.194.34	255.255.255.252	eth1
51.37.194.44	51.37.194.38	255.255.255.252	eth2
51.37.194.48	51.37.194.38	255.255.255.252	eth2

- n11

Destination	Gateway	GenMask	Iface
51.37.192.0	51.37.194.46	255.255.255.0	aaa
51.37.193.0	51.37.194.37	255.255.255.128	eth0
51.37.193.128	51.37.194.45	255.255.255.128	eth0
51.37.194.0	51.37.194.45	255.255.255.224	eth2
51.37.194.32	51.37.194.37	255.255.255.252	eth1
51.37.194.36	0.0.0.0	255.255.255.252	eth2
51.37.194.40	51.37.194.45	255.255.255.252	eth1
51.37.194.44	0.0.0.0	255.255.255.252	eth2
51.37.194.48	51.37.194.45	255.255.255.252	eth2

- n2

Destination	Gateway	GenMask	Iface
51.37.192.0	51.37.194.42	255.255.255.0	aaa
51.37.193.0	51.37.194.33	255.255.255.128	eth0
51.37.193.128	51.37.194.42	255.255.255.128	eth0
51.37.194.0	51.37.194.42	255.255.255.224	eth2
51.37.194.32	0.0.0.0	255.255.255.252	eth1
51.37.194.36	51.37.194.33	255.255.255.252	eth2
51.37.194.40	0.0.0.0	255.255.255.252	eth1
51.37.194.44	51.37.194.42	255.255.255.252	eth2
51.37.194.48	51.37.194.42	255.255.255.252	eth2

- n6

Destination	Gateway	GenMask	Iface
51.37.192.0	51.37.194.50	255.255.255.0	aaa
51.37.193.0	51.37.194.41	255.255.255.128	eth0
51.37.193.128	51.37.194.42	255.255.255.128	eth0
51.37.194.0	51.37.194.50	255.255.255.224	eth2
51.37.194.32	51.37.194.41	255.255.255.252	eth1
51.37.194.36	51.37.194.46	255.255.255.252	eth2
51.37.194.40	0.0.0.0	255.255.255.252	eth1
51.37.194.44	0.0.0.0	255.255.255.252	eth2
51.37.194.48	0.0.0.0	255.255.255.252	eth2

- n3

Destination	Gateway	GenMask	Iface
51.37.192.0	0.0.0.0	255.255.255.0	aaa
51.37.193.0	51.37.194.49	255.255.255.128	eth0
51.37.193.128	0.0.0.0	255.255.255.128	eth0
51.37.194.0	0.0.0.0	255.255.255.224	eth2
51.37.194.32	51.37.194.49	255.255.255.252	eth1
51.37.194.36	51.37.194.49	255.255.255.252	eth2
51.37.194.40	51.37.194.49	255.255.255.252	eth1
51.37.194.44	51.37.194.49	255.255.255.252	eth2
51.37.194.48	0.0.0.0	255.255.255.252	eth2

- n12

Destination	Gateway	GenMask	Iface
0.0.0.0	51.37.192.1	0.0.0.0	eth0

- n10

Destination	Gateway	GenMask	Iface
0.0.0.0	51.37.192.1	0.0.0.0	eth0

- n15

Destination	Gateway	GenMask	Iface
0.0.0.0	51.37.194.1	0.0.0.0	eth0

- dhcp-server

Destination	Gateway	GenMask	Iface
0.0.0.0	51.37.194.1	0.0.0.0	eth0

- n7

Destination	Gateway	GenMask	Iface
0.0.0.0	51.37.193.129	0.0.0.0	eth0

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

```
root@n8:/tmp/pycore.59710/n8.conf# ping -c3 51.37.194.2
PING 51.37.194.2 (51.37.194.2) 56(84) bytes of data.
64 bytes from 51.37.194.2: icmp_seq=1 ttl=60 time=0.285 ms
64 bytes from 51.37.194.2: icmp_seq=2 ttl=60 time=0.285 ms
64 bytes from 51.37.194.2: icmp_seq=3 ttl=60 time=0.281 ms

--- 51.37.194.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.281/0.283/0.285/0.019 ms
root@n8:/tmp/pycore.59710/n8.conf#
```

Figura 2: PING entre red A y red B

```
root@n8:/tmp/pycore.59710/n8.conf# ping -c3 51.37.192.2
PING 51.37.192.2 (51.37.192.2) 56(84) bytes of data.
64 bytes from 51.37.192.2: icmp_seq=1 ttl=60 time=0.412 ms
64 bytes from 51.37.192.2: icmp_seq=2 ttl=60 time=0.273 ms
64 bytes from 51.37.192.2: icmp_seq=3 ttl=60 time=0.239 ms

--- 51.37.192.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.239/0.308/0.412/0.074 ms
root@n8:/tmp/pycore.59710/n8.conf# ~
```

Figura 3: PING entre red A y red X

```
root@n8:/tmp/pycore.59710/n8.conf# ping -c3 51.37.193.130
PING 51.37.193.130 (51.37.193.130) 56(84) bytes of data.
64 bytes from 51.37.193.130: icmp_seq=1 ttl=60 time=0.231 ms
64 bytes from 51.37.193.130: icmp_seq=2 ttl=60 time=0.289 ms
64 bytes from 51.37.193.130: icmp_seq=3 ttl=60 time=0.290 ms

--- 51.37.193.130 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.231/0.270/0.290/0.027 ms
root@n8:/tmp/pycore.59710/n8.conf#
```

Figura 4: PING entre red A y red Y

```
root@n15:/tmp/pycore.59710/n15.conf# ping -c3 51.37.192.2
PING 51.37.192.2 (51.37.192.2) 56(84) bytes of data.
64 bytes from 51.37.192.2: icmp_seq=1 ttl=63 time=0.171 ms
64 bytes from 51.37.192.2: icmp_seq=2 ttl=63 time=0.128 ms
64 bytes from 51.37.192.2: icmp_seq=3 ttl=63 time=0.150 ms

--- 51.37.192.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.128/0.149/0.171/0.022 ms
root@n15:/tmp/pycore.59710/n15.conf#
```

Figura 5: PING entre red B y red X

```
root@n15:/tmp/pycore.59710/n15.conf# ping -c3 51.37.193.130
PING 51.37.193.130 (51.37.193.130) 56(84) bytes of data.
64 bytes from 51.37.193.130: icmp_seq=1 ttl=63 time=0.208 ms
64 bytes from 51.37.193.130: icmp_seq=2 ttl=63 time=0.166 ms
64 bytes from 51.37.193.130: icmp_seq=3 ttl=63 time=0.209 ms

--- 51.37.193.130 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.166/0.194/0.209/0.023 ms
root@n15:/tmp/pycore.59710/n15.conf#
```

Figura 6: PING entre red B y red Y

```
root@n10:/tmp/pycore.59710/n10.conf# ping -c3 51.37.193.130
PING 51.37.193.130 (51.37.193.130) 56(84) bytes of data.
64 bytes from 51.37.193.130: icmp_seq=1 ttl=63 time=0.184 ms
64 bytes from 51.37.193.130: icmp_seq=2 ttl=63 time=0.140 ms
64 bytes from 51.37.193.130: icmp_seq=3 ttl=63 time=0.181 ms

--- 51.37.193.130 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.140/0.168/0.184/0.022 ms
root@n10:/tmp/pycore.59710/n10.conf#
```

Figura 7: PING entre red X y red Y

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

2.1. Utilizando el comando traceroute(1)/mtr(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?

```
root@n8:/tmp/pycore.43148/n8.conf# traceroute -U 51.37.192.2
traceroute to 51.37.192.2 (51.37.192.2), 30 hops max, 60 byte packets
 1  51.37.193.1 (51.37.193.1)  0.169 ms  0.058 ms  0.053 ms
 2  51.37.194.34 (51.37.194.34)  0.101 ms  0.150 ms  0.080 ms
 3  51.37.194.42 (51.37.194.42)  0.121 ms  0.103 ms  0.100 ms
 4  51.37.194.50 (51.37.194.50)  0.146 ms  0.127 ms  0.129 ms
 5  51.37.192.2 (51.37.192.2)  0.175 ms  0.231 ms  0.157 ms
root@n8:/tmp/pycore.43148/n8.conf#
```

Figura 8: Traceroute usando UDP entre n8 y n10

```
root@n8:/tmp/pycore.43148/n8.conf# traceroute -I 51.37.192.2
traceroute to 51.37.192.2 (51.37.192.2), 30 hops max, 60 byte packets
 1  51.37.193.1 (51.37.193.1)  0.163 ms  0.045 ms *
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * 51.37.192.2 (51.37.192.2)  0.215 ms  0.075 ms
root@n8:/tmp/pycore.43148/n8.conf#
```

Figura 9: Traceroute usando ICMP entre n8 y n10

```
root@n8:/tmp/pycore.42299/n8.conf# tcpdump -i eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:05:04.521042 IP 51.37.193.2.33996 > 51.37.193.130.33434: UDP, length 32
22:05:04.521146 IP 51.37.193.1 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.521207 IP 51.37.193.2.52457 > 51.37.193.130.33435: UDP, length 32
22:05:04.521247 IP 51.37.193.1 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.521328 IP 51.37.193.2.41143 > 51.37.193.130.33436: UDP, length 32
22:05:04.521369 IP 51.37.193.1 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.521414 IP 51.37.193.2.54664 > 51.37.193.130.33437: UDP, length 32
22:05:04.521495 IP 51.37.194.38 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.521541 IP 51.37.193.2.46882 > 51.37.193.130.33438: UDP, length 32
22:05:04.521599 IP 51.37.194.38 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.521641 IP 51.37.193.2.40312 > 51.37.193.130.33439: UDP, length 32
22:05:04.521698 IP 51.37.194.38 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.521792 IP 51.37.193.2.36273 > 51.37.193.130.33440: UDP, length 32
22:05:04.521909 IP 51.37.194.42 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.521958 IP 51.37.193.2.46687 > 51.37.193.130.33441: UDP, length 32
22:05:04.522038 IP 51.37.194.42 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.522081 IP 51.37.193.2.37087 > 51.37.193.130.33442: UDP, length 32
22:05:04.522158 IP 51.37.194.42 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.522200 IP 51.37.193.2.45689 > 51.37.193.130.33443: UDP, length 32
^C22:05:04.522315 IP 51.37.194.50 > 51.37.193.2: ICMP time exceeded in-transit, length 68
22:05:04.522360 IP 51.37.193.2.37779 > 51.37.193.130.33444: UDP, length 32
```

Figura 10: Como puede observarse, usando ICMP devuelve *ICMP time exceeded*

Aplicando traceroute –icmp solo nos muestra el primer salto y la llegada al host destino. Aplicando traceroute –udp nos muestra todos los saltos intermedios hasta llegar al host

destino.

La diferencia radica en la información que viene en el paquete de vuelta: en el caso de UDP el TTL vuelve expirado y en ICMP, HOST Unreachable.

En traceroute UDP, el cliente transmite a UDP un simple paquete a un valor de puerto de destino. Si el puerto es filtrado, devuelve HOST Unreachable. Por eso es que para estos casos es mejor usar ICMP.

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

```
root@n8:/tmp/pycore.43148/n8.conf# ping -c3 51.37.193.130
PING 51.37.193.130 (51.37.193.130) 56(84) bytes of data.
64 bytes from 51.37.193.130: icmp_seq=1 ttl=60 time=0.356 ms
64 bytes from 51.37.193.130: icmp_seq=2 ttl=60 time=0.288 ms
64 bytes from 51.37.193.130: icmp_seq=3 ttl=60 time=0.249 ms

--- 51.37.193.130 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.249/0.297/0.356/0.048 ms
root@n8:/tmp/pycore.43148/n8.conf#
```

Figura 11: PING entre n8 y n5

El valor inicial del TTL es 60.

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

```
root@n1:/tmp/pycore.43148/n1.conf# tcpdump -i eth0 -v
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
15:33:35.891932 IP (tos 0x0, ttl 64, id 2175, offset 0, flags [DF], proto ICMP (1), length 84)
    51.37.193.2 > 51.37.193.130: ICMP echo request, id 37, seq 1, length 64
15:33:35.892141 IP (tos 0x0, ttl 60, id 5736, offset 0, flags [none], proto ICMP (1), length 84)
    51.37.193.130 > 51.37.193.2: ICMP echo reply, id 37, seq 1, length 64
15:33:36.890896 IP (tos 0x0, ttl 64, id 2208, offset 0, flags [DF], proto ICMP (1), length 84)
    51.37.193.2 > 51.37.193.130: ICMP echo request, id 37, seq 2, length 64
15:33:36.891064 IP (tos 0x0, ttl 60, id 5821, offset 0, flags [none], proto ICMP (1), length 84)
    51.37.193.130 > 51.37.193.2: ICMP echo reply, id 37, seq 2, length 64
15:33:37.891747 IP (tos 0x0, ttl 64, id 2353, offset 0, flags [DF], proto ICMP (1), length 84)
    51.37.193.2 > 51.37.193.130: ICMP echo request, id 37, seq 3, length 64
15:33:37.891956 IP (tos 0x0, ttl 60, id 5830, offset 0, flags [none], proto ICMP (1), length 84)
    51.37.193.130 > 51.37.193.2: ICMP echo reply, id 37, seq 3, length 64
```

Figura 12: TCPdump que captura tráfico en la interfaz eth0

```
root@n1:/tmp/pycore.43148/n1.conf# tcpdump -i eth2 -v
tcpdump: listening on eth2, link-type EN10MB (Ethernet), capture size 262144 bytes
15:33:35.891967 IP (tos 0x0, ttl 63, id 2175, offset 0, flags [DF], proto ICMP (1), length 84)
    51.37.193.2 > 51.37.193.130: ICMP echo request, id 37, seq 1, length 64
15:33:36.890924 IP (tos 0x0, ttl 63, id 2208, offset 0, flags [DF], proto ICMP (1), length 84)
    51.37.193.2 > 51.37.193.130: ICMP echo request, id 37, seq 2, length 64
15:33:37.891782 IP (tos 0x0, ttl 63, id 2353, offset 0, flags [DF], proto ICMP (1), length 84)
    51.37.193.2 > 51.37.193.130: ICMP echo request, id 37, seq 3, length 64
^C
3 packets captured
3 packets received by filter
0 packets dropped by kernel
root@n1:/tmp/pycore.43148/n1.conf#
```

Figura 13: TCPdump que captura tráfico en la interfaz eth2

Se puede observar en las diferentes capturas que por cada salto que hace el TTL se decrementa en 1: empieza en 64, luego hace un salto y se decrementa en 1 y pasa a 63. El TTL se decrementa tantas veces como hosts haya pasado.

2.4. 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?

```
root@n8:/tmp/pycore.43148/n8.conf# hping3 -t 1 51.37.193.130
HPING 51.37.193.130 (eth0 51.37.193.130): NO FLAGS are set, 40 headers + 0 data bytes
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
TTL 0 during transit from ip=51.37.193.1 name=UNKNOWN
^C
--- 51.37.193.130 hping statistic ---
18 packets transmitted, 9 packets received, 50% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@n8:/tmp/pycore.43148/n8.conf#
```

Figura 14: Envío del datagrama desde n8 a n7 usando hping3

La respuesta nos dice que el TTL dio 0 antes de llegar al host(que es desconocido). Esto es porque el TTL se va decrementando en 1 cada vez que pasa por un host(router) y como lo configuramos en 1, el primer router lo decremento, este TTL quedó en 0 y por ende, el router lo devuelve con "tiempo de vida agotado".

3. Fragmentación:

3.1. Cambiando los MTU y enviando tráfico desde un host de la parte "A" a la "B" ver de forzar que los routers fragmenten.

Si bien cambiamos el mtu del host de la red B con el comando *ip link set dev eth0 mtu 500* y corrimos el ping mientras capturábamos el tráfico con wireshark, nunca llegamos a obtener un paquete fragmentado.

También intentamos modificar el MTU de un router, pero tampoco funcionó(el ping nos decia "Host Unreachable").

También probé, en la máquina virtual fuera de CORE(simplemente por curiosidad), esos comandos pero con un MTU más bajo y se rompió la máquina virtual(CORE dejó de funcionar y se cortó la conexión a internet).

Pistas de que, probablemente, íbamos en camino(o no):

- Había un flag(don't fragment) que aparecía seteado. Buscamos qué indica ese flag y parece ser que, de estar seteado, no permite a nadie fragmentar. Esto significaría que estuvimos forzando a fragmentar cuando, en realidad, nunca fue una opción.
- Para comprobar que hubiera fragmentación, filtramos en el wireshark según el flag MF para buscar aquel paquete que tuviera ese bit seteado. En ningún momento encontramos un paquete con tal bit seteado.

3.2. ¿Dónde se fragmenta y donde se re-ensambla? ¿Qué campos son cambiados durante la fragmentación? ¿Cómo se identifican los fragmentos de un mismo segmento?

La fragmentación puede tener lugar en el emisor inicial o en los routers que están entre el emisor y el receptor. Si un datagrama es fragmentado, no será ensamblado(desfragmentado) de nuevo hasta llegar al receptor. Si es necesario, un paquete ya fragmentado puede ser fragmentado otra vez (por ejemplo durante un cambio de método de transmisión).

Cada fragmento del datagrama original obtiene en vez del datagram header (cabecera de datagrama) del paquete original un denominado fragment header (cabecera de fragmento) que contiene entre otras cosas el offset que indica la porción de datos enviado en este paquete en relación al paquete original. El fragment offset (13 bit en el IP header) está indicado en bloques de 64 bits. Todos los fragmentos menos el último tienen el more fragments flag con valor "1". El campo de longitud en el IP header contiene la longitud del fragmento, y se calcula el checksum para cada fragmento apartadamente, mientras que el resto del header corresponde al header original.

El receptor es el responsable de reensamblar todos los fragmentos en el orden correcto para obtener el datagrama original y entregarlos al protocolo de nivel superior. El reensamblaje se espera que ocurra en el equipo receptor, pero en la práctica puede ocurrir también en routers intermedios.

3.3. ¿Considera que la fragmentación es necesaria o que puede ser evitada? Justfique.

Debería ser evitada a toda costa por dos grandes razones:

- La primera, que puede tener una gran influencia negativa en la actuación y en el flujo de datos.
- La segunda, que si se pierde un fragmento hay que retransmitir nuevamente los del paquete original. Sin embargo, IP no tienen mecanismos de seguridad o de timeout y es dependiente de las funciones de seguridad que implementen los protocolos de las capas más altas tales como TCP.

En cuanto a si es necesaria o no, quizás podría evitarse si se controlara el tamaño de los paquetes que se envían. La fragmentación es necesaria cuando el host emisor tiene un MTU muy superior al de su receptor y los paquetes que envía son muy grandes. Si se controla ese aspecto, entonces ésta puede evitarse.