

Tarea 01 - Mininet

Sección 2

Jazmín Jassive Cuitiño Mendoza
e-mail: jazmin.cuitino@mail.udp.cl

Abril de 2024

Índice

1. Introducción y Equipo	2
2. Desarrollo de las actividades	2
2.1. Actividad 1:	2
2.1.1. Información de cada host	5
2.1.2. Pings (h1,h2)	7
2.1.3. Pings (h1,h3)	9
2.1.4. Pings (h3,h4)	10
2.2. Actividad 2:	11
2.3. Actividad 3:	13
2.4. Actividad 4:	17
2.4.1. Paso I: Comando iperf	19
2.4.2. Paso II: Transferencia de archivo usando FTP	21
3. Análisis de resultados	24
4. Conclusiones y comentarios	26

1. Introducción y Equipo

EL presente informe tiene como objetivo documentar una serie de actividades destinadas a trabajar y comprender la plataforma de emulación de redes conocida como **Mininet**. Dentro de las actividades a realizar, esta la construcción de redes virtuales, utilización de SDN, ping entre host's, y análisis de los paquetes vía wireshark.

Características del equipo:

- **Procesador:** Intel Core i7 (10° Generación)
- **Sistema Operativo:** Kali Linux (Nativo)

La tarea no se realizara con la VM de mininet, si no que se instala todo localmente.

2. Desarrollo de las actividades

2.1. Actividad 1:

La primera tarea a realizar implica la creación de una red compuesta por **4 Switches** y **4 Hosts**. Visualmente, se la red diseñada sera de la siguiente manera:

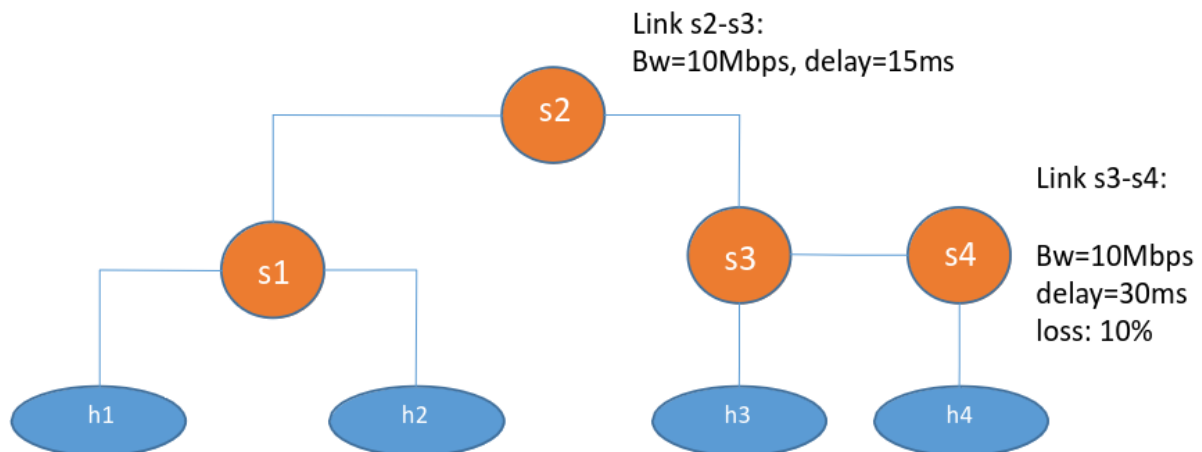


Figura 1: Red a diseñar

Se aprecia entonces, de la imagen anterior, que entre los enlaces existentes solamente 2 de ellos requieren configuraciones adicionales:

1. s2 -> s3:

- **BW:** 10Mbps

- **delay:** 15ms
2. **s3 -> s4:**
- **BW:** 10Mbps
 - **delay:** 30ms
 - **loss:** 10

Para conseguir las características específicas de los links solicitados es que se diseña el siguiente código:

```
from mininet.topo import Topo

class Act_1(Topo):
    "Topologia Simple"

    def build(self):
        # Añadiendo Hosts
        h1 = self.addHost('h1')
        h2 = self.addHost('h2')
        h3 = self.addHost('h3')
        h4 = self.addHost('h4')

        # Añadiendo Switchs
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')

        # Añadiendo Links
        self.addLink(h1, s1)
        self.addLink(h2, s1)
        self.addLink(s1, s2)
        self.addLink(s2, s3, bw=10, delay="15ms")
        self.addLink(s3, s4, bw=10, delay="30ms", loss=10)
        self.addLink(h3, s3)
        self.addLink(h4, s4)

topos = { "topo": ( lambda : Act_1() ) }
```

En el código precedente, se puede apreciar que se hace uso de la biblioteca **mininet** para hacer uso de sus funcionalidades, en particular, se importa la clase **Topo**. De esta forma, se define una clase llamada **Act_1**, la que hereda de la clase **topo**, y que contendrá el diseño de nuestra red personalizada.

Una vez instanciadas las clases necesarias, se continua añadiendo en primer lugar los hosts con la función **addHost()**, y luego los Switchs, mediante la funcion **addSwitch()**.

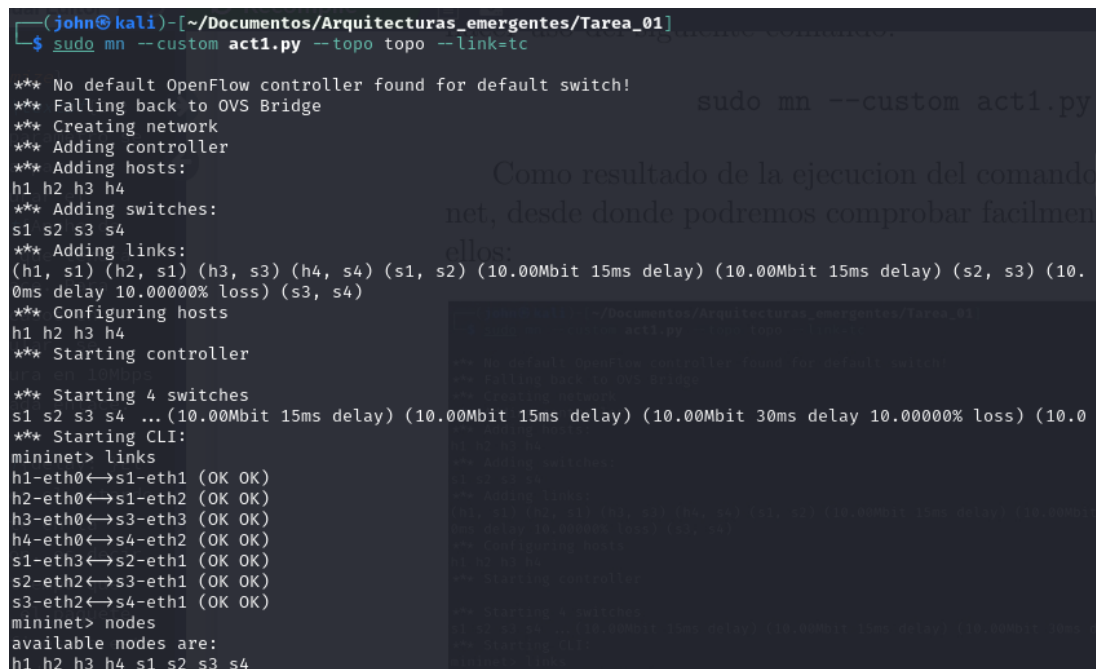
Habiendo añadido entonces los equipos necesarios, se procede a configurar las conexiones entre ellos, para esto se utiliza la función **addLinks()**. De todas las conexiones, las únicas que necesitan configuracion extra son la conexión (s2,s3) y la conexión (s3, s4), las que requieren valores de **bw**, **delay** y **loss** específicos. A continuacion, se describen brevemente estos parametros:

- **BW:** Este parametro se utiliza para configurar el *Ancho de banda* que tendrá el enlace. Para estos casos en particular, se configura en 10Mbps para cada enlace.
- **delay:** El delay es el retardo presente en la conexión, es decir, es el tiempo que demora el paquete (para fines de esta actividad, paquetes ICMP) entre el dispositivo de origen y destino.
- **Loss:** Es la perdida de paquetes, es decir, es el porcentaje de paquetes que se perderán, que no llegaran al destino. En este caso, se configura en 10 % entre s3 y s4.

Para ejecutar este script, levantar la red diseñada y continuar con la actividad, se debe hacer uso del siguiente comando:

```
sudo mn --custom act1.py --topo topo --link=tc
```

Como resultado de la ejecución del comando precedente, accedemos a la consola de mininet, desde donde podremos comprobar facilmente los nodos existentes y las conexiones entre ellos:



```
(john@kali)-[~/Documentos/Arquitecturas_emergentes/Tarea_01]
$ sudo mn --custom act1.py --topo topo --link=tc

*** No default OpenFlow controller found for default switch!
*** Falling back to OVS Bridge
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s1) (h3, s3) (h4, s4) (s1, s2) (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (s2, s3) (10.00Mbit 30ms delay 10.00000% loss) (s3, s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
*** Starting 4 switches
s1 s2 s3 s4 ... (10.00Mbit 15ms delay) (10.00Mbit 15ms delay) (10.00Mbit 30ms delay 10.00000% loss) (10.00Mbit 30ms delay 10.00000% loss)
*** Starting CLI:
mininet> links
h1-eth0<->s1-eth1 (OK OK)
h2-eth0<->s1-eth2 (OK OK)
h3-eth0<->s3-eth3 (OK OK)
h4-eth0<->s4-eth2 (OK OK)
s1-eth3<->s2-eth1 (OK OK)
s2-eth2<->s3-eth1 (OK OK)
s3-eth2<->s4-eth1 (OK OK)
mininet> nodes
available nodes are:
h1 h2 h3 h4 s1 s2 s3 s4
```

Figura 2: Levantamiento red mininet

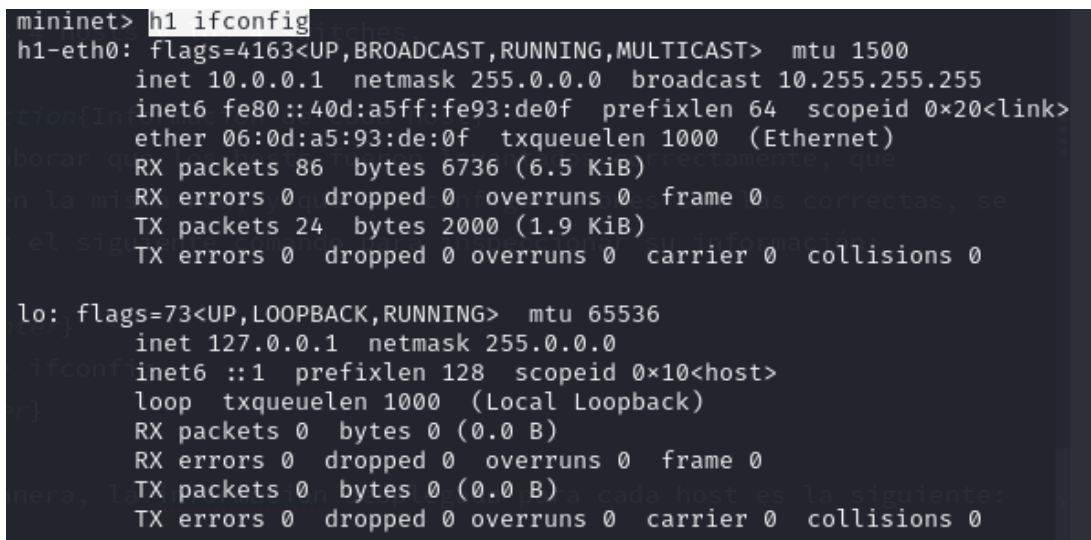
Se aprecia de esta forma que las conexiones fueron hechas correctamente, pues al utilizar el comando **links** se despliegan todos los enlaces existentes en la red, junto con sus estados. Además, con el comando **nodes**, se pueden ver los nodos activos dentro de la red, es decir, los 4 hosts y los 4 switches.

2.1.1. Información de cada host

Para corroborar que los hosts fueron levantados correctamente, que quedaron en la misma red, y que sus configuraciones son las correctas, se puede usar el siguiente comando para inspeccionar su información:

```
<host> ifconfig
```

De esta manera, la información desplegada para cada host es la siguiente:



```
mininet> h1 ifconfig
h1-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.0.0.1 netmask 255.0.0.0 broadcast 10.255.255.255
        inet6 fe80::40d:a5ff:fe93:de0f prefixlen 64 scopeid 0x20<link>
        ether 06:0d:a5:93:de:0f txqueuelen 1000 (Ethernet)
        RX packets 86 bytes 6736 (6.5 KiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 24 bytes 2000 (1.9 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
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1000 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 3: Información del host h1

```
mininet> h2 ifconfig
h2-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.2 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::f0e8:64ff:fea7:1d92 prefixlen 64 scopeid 0x20<link>
    ether f2:e8:64:a7:1d:92 txqueuelen 1000 (Ethernet)
    RX packets 92 bytes 7156 (6.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 25 bytes 2070 (2.0 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
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 4: Información del host h2

```
mininet> h3 ifconfig
h3-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.3 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::50a5:b6ff:fed4:e959 prefixlen 64 scopeid 0x20<link>
    ether 52:a5:b6:d4:e9:59 txqueuelen 1000 (Ethernet)
    RX packets 82 bytes 6224 (6.0 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 13 bytes 1006 (1006.0 B)
    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
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
```

Figura 5: Información del host h3

```
mininet> h4 ifconfig
h4-eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.0.4 netmask 255.0.0.0 broadcast 10.255.255.255
    inet6 fe80::7462:65ff:fe3a:1d6f prefixlen 64 scopeid 0<link>
    ether 76:62:65:3a:1d:6f txqueuelen 1000 (Ethernet)
    RX packets 74 bytes 5648 (5.5 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 13 bytes 1006 (1006.0 B)
    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
    inet6 ::1 prefixlen 128 scopeid 0<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 6: Información del host h4

Adicionalmente, también se pueden verificar que los puertos locales de los Switches estén correctamente configurados para traficar información, eso se hace de la siguiente manera:

```
mininet> dpctl dump-ports s1
*** s1 ***
OFPST_PORT reply (xid=0x4): 1 ports
  port LOCAL: rx pkts=0, bytes=0, drop=83, errs=0, frame=0, over=0, crc=0
               tx pkts=0, bytes=0, drop=0, errs=0, coll=0
```

Figura 7: Puertos S1

Luego de realizar estos pasos, se puede asegurar que la configuración fue correcta, y no debe haber problema al comunicar los distintos hosts, para verificar esto es que se realizan las siguientes actividades.

2.1.2. Pings (h1,h2)

Para comprobar el correcto funcionamiento y configuración de la red diseñada, se enviarán 10 pings desde el host1 al host2, de la siguiente manera:

```

mininet> h1 ping -c 10 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
 64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.436 ms
 64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.092 ms
 64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.105 ms
 64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.089 ms
 64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.099 ms
 64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.095 ms
 64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.092 ms
 64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.092 ms
 64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.096 ms
 64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.100 ms

— 10.0.0.2 ping statistics —
10 packets transmitted, 10 received, 0% packet loss, time 9213ms
rtt min/avg/max/mdev = 0.089/0.129/0.436/0.102 ms
mininet>

```

Figura 8: 10 pings entre h1 y h2

Se aprecia entonces que fueron enviados y recibidos correctamente estos paquetes ICMP, obteniendo de esta manera un 0% de packet loss, o en otras palabras, 100% de intervalo de confianza. Un dato importante es el tiempo que transcurrió desde el envío hasta la recepción de paquetes, que en promedio, estuvo entre los 0.09 y 0.1 ms; esto debido a que en los enlaces entre los host y el switch no había delay.

Ahora bien, al observar el tráfico en esta red con **Wireshark**, se observa lo siguiente:

12 18.772255616	06:0d:a5:93:de:0f	Broadcast	ARP	42 Who has 10.0.0.2? Tell 10.0.0.1
13 18.772541194	f2:e8:64:a7:1d:92	06:0d:a5:93:de:0f	ARP	42 10.0.0.2 is at f2:e8:64:a7:1d:92
14 18.772546844	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=1/256, ttl=64 (reply in 15)
15 18.772670314	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=1/256, ttl=64 (request in 14)
16 18.928969993	fe80::a8a1:bcff:feb...	ff02::2	ICMPv6	70 Router Solicitation from aa:a1:bc:b2:c4:79
17 18.929454124	fe80::f0e8:64ff:fea...	ff02::2	ICMPv6	70 Router Solicitation from f2:e8:64:a7:1d:92
18 18.974963860	fe80::7462:65ff:fe3...	ff02::2	ICMPv6	70 Router Solicitation from 76:62:65:3a:1d:6f
19 19.793007342	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=2/512, ttl=64 (reply in 20)
20 19.793059434	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=2/512, ttl=64 (request in 19)
21 20.817023460	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=3/768, ttl=64 (reply in 22)
22 20.817075892	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=3/768, ttl=64 (request in 21)
23 20.976960857	fe80::40d:a5ff:fe93...	ff02::2	ICMPv6	70 Router Solicitation from 06:0d:a5:93:de:0f
24 21.840944063	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=4/1024, ttl=64 (reply in 25)
25 21.840994368	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=4/1024, ttl=64 (request in 24)
26 22.864962453	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=5/1280, ttl=64 (reply in 27)
27 22.865016686	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=5/1280, ttl=64 (request in 26)
28 23.792905232	f2:e8:64:a7:1d:92	06:0d:a5:93:de:0f	ARP	42 Who has 10.0.0.1? Tell 10.0.0.2
29 23.792915399	06:0d:a5:93:de:0f	f2:e8:64:a7:1d:92	ARP	42 10.0.0.1 is at 06:0d:a5:93:de:0f
30 23.889003354	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=6/1536, ttl=64 (reply in 31)
31 23.889056062	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=6/1536, ttl=64 (request in 30)
32 24.913060192	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=7/1792, ttl=64 (reply in 33)
33 24.913109977	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=7/1792, ttl=64 (request in 32)
34 25.936975736	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=8/2048, ttl=64 (reply in 35)
35 25.937026783	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=8/2048, ttl=64 (request in 34)
36 26.960995302	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=9/2304, ttl=64 (reply in 37)
37 26.961048109	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=9/2304, ttl=64 (request in 36)
38 27.985030025	10.0.0.1	10.0.0.2	ICMP	98 Echo (ping) request id=0x67f2, seq=10/2560, ttl=64 (reply in 39)
39 27.985084016	10.0.0.2	10.0.0.1	ICMP	98 Echo (ping) reply id=0x67f2, seq=10/2560, ttl=64 (request in 38)

Figura 9: Captura de paquetes comunicación h1,h2

De la comunicación entre h1 y h2 podemos distinguir los siguientes protocolos:

- **ARP:** Protocolo de comunicaciones de la capa de enlace de datos, responsable de encontrar la dirección de hardware (Ethernet MAC) que corresponde a una determinada dirección IP.

- **ICMP:** Es el protocolo utilizado para realizar los pings. Se usa tanto para el paquete de ida, como para el paquete de respuesta.
- **ICMPv6:** Este es un protocolo de red utilizado en redes IPv6 para el intercambio de mensajes de control y error entre dispositivos de red. Su objetivo es detectar errores en los paquetes, además de mantenimiento y diagnostico en la capa de internet. En este caso, los paquetes son “Router Solicitation” (Solicitud de enrutador), lo que significa que un dispositivo está solicitando información de configuración a un enrutador en la red.

2.1.3. Pings (h1,h3)

Ahora bien, ya se hicieron los pings entre dos host en los que sus enlaces no presentaban ningún tipo de delay. Para esta ocasión, se realiza la comunicación entre h1 y h3, la que si contiene un delay en el enlace (s2,s3) de 15ms; esto implica que la respuesta de los pings ya no sera del orden de 0.09 ms, si no que se debe multiplicar x2 los 15ms de retraso (los tiempos de viaje de ida y vuelta sumados), obteniendo un promedio de 30,3ms de respuesta para cada ping:

```
mininet> h1 ping -c 10 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=31.4 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=30.2 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=30.3 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=30.3 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=30.3 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=30.3 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=30.3 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=30.3 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=30.2 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=30.3 ms

— 10.0.0.3 ping statistics —
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 30.176/30.377/31.419/0.348 ms
```

Figura 10: 10 pings entre h1 y h3

De la mano con este envío de pings, se encuentra también la captura de wireshark, en donde se aprecian los paquetes ICMP involucrados en esta comunicación, además de los paquetes pertenecientes a los protocolo ARP e ICMPv6 ya detallados anteriormente.

1 0.000000000	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=1/256, ttl=64 (reply in 2)
2 0.000051601	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=1/256, ttl=64 (request in 1)
3 1.0006666193	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=2/512, ttl=64 (reply in 4)
4 1.000709596	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=2/512, ttl=64 (request in 3)
5 2.0020066898	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=3/768, ttl=64 (reply in 6)
6 2.002049822	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=3/768, ttl=64 (request in 5)
7 3.003529709	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=4/1024, ttl=64 (reply in 8)
8 3.003575170	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=4/1024, ttl=64 (request in 7)
9 4.005010508	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=5/1280, ttl=64 (reply in 10)
10 4.005056921	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=5/1280, ttl=64 (request in 9)
11 5.006456338	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=6/1536, ttl=64 (reply in 12)
12 5.006501225	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=6/1536, ttl=64 (request in 11)
13 6.007912651	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=7/1792, ttl=64 (reply in 14)
14 6.007962464	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=7/1792, ttl=64 (request in 13)
15 7.009415635	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=8/2048, ttl=64 (reply in 16)
16 7.009462652	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=8/2048, ttl=64 (request in 15)
17 8.010871772	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=9/2304, ttl=64 (reply in 18)
18 8.010928255	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=9/2304, ttl=64 (request in 17)
19 9.012312533	10.0.0.1	10.0.0.3	ICMP	98 Echo (ping) request	id=0xe4c3, seq=10/2560, ttl=64 (reply in 20)
20 9.012355653	10.0.0.3	10.0.0.1	ICMP	98 Echo (ping) reply	id=0xe4c3, seq=10/2560, ttl=64 (request in 19)
21 11.085524505	52:a5:b6:d4:e9:59	06:0d:a5:93:de:0f	ARP	42 Who has 10.0.0.1? Tell 10.0.0.3	
22 11.117329143	06:0d:a5:93:de:0f	52:a5:b6:d4:e9:59	ARP	42 10.0.0.1 is at 06:0d:a5:93:de:0f	
23 42.589520303	fe80::f0e8:64ff:fea...	ff02::2	ICMPv6	70 Router Solicitation from f2:e8:64:a7:1d:92	

Figura 11: Captura de paquetes comunicación h1,h2

2.1.4. Pings (h3,h4)

En este punto se solicita determinar la cantidad de pings necesaria para conseguir un 5 % de packet loss, sin embargo, al estar configurada la red con un 10 % de loss en el enlace entre s3 y s4, no fue posible bajar de este valor, y a medida que se aumentaron la cantidad de pings, incluso empeoro el packet loss. El valor mas pequeño de packet loss obtenido fue un 10 %, cuando se enviaron apenas 20 pings. La evidencia a continuación:

```
--- 10.0.0.4 ping statistics ---
20 packets transmitted, 18 received, 10% packet loss, time 19042ms
rtt min/avg/max/mdev = 60.487/61.628/66.533/1.384 ms
mininet> █
```

Figura 12: 20 Pings h3 - h4

```
64 bytes from 10.0.0.4: icmp_seq=798 ttl=64 time=61.2 ms
64 bytes from 10.0.0.4: icmp_seq=799 ttl=64 time=60.5 ms

--- 10.0.0.4 ping statistics ---
800 packets transmitted, 639 received, 20.125% packet loss, time 802659ms
rtt min/avg/max/mdev = 60.257/61.390/64.683/0.631 ms
mininet> █
```

Figura 13: 800 Pings h3 - h4

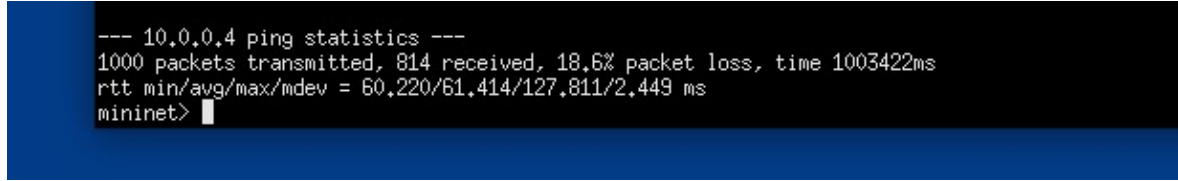
A screenshot of a terminal window with a dark background and a blue border on the left. The text displayed is: --- 10.0.0.4 ping statistics ---
1000 packets transmitted, 814 received, 18.6% packet loss, time 1003422ms
rtt min/avg/max/mdev = 60.220/61.414/127.811/2.449 ms
mininet> |

Figura 14: 1000 Pings h3 - h4

2.2. Actividad 2:

Para esta actividad es necesario montar un servidor http en el host1. Para esto, se utiliza el comando:

```
h1 python -m http.server 80 &
```

De esta forma, se indica en mininet que se debe levantar el servidor http en el puerto 80 en el host h1. Posteriormente, y luego de haber levantado este servidor, se solicita hacer una llamada tipo GET desde el host2 (h2) hacia el servidor recién levantado en h1, para esto se hace uso del siguiente comando:

```
h2 wget -o h1
```

Al ejecutar estas instrucciones se obtiene la siguiente respuesta desde la consola en mininet:

```

mininet> h1 python -m http.server 80 &
mininet> h2 wget -O - h1
--2024-03-30 21:17:03-- http://10.0.0.1/31a:73
Conectando con 10.0.0.1:80... conectado.1f:03:6f
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 518 [text/html] from 82:67:9f:f2:c3:09
Grabando a: «STDOUT» from 3e:6f:ee:88:c5:89
- Router Solicitation from da:c3:a5:42:67:00
- 0%[====>] 0 --.-KB/s <!DOCTYPE HTML>
<html lang="en"> Solicitation from 32:0a:df:7a:94:8e
<head> Solicitation from d2:0d:8b:5a:84:44
<meta charset="utf-8"> from 66:93:f1:21:1a:73
<title>Directory listing for /</title>
</head>
<body>1-eth1, id 0
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="act1.py">act1.py</a></li>
<li><a href="act_1.py">act_1.py</a></li>
<li><a href="Guia_mininet.pdf">Guia_mininet.pdf</a></li>
<li><a href="h1_ping_h2.jpg">h1_ping_h2.jpg</a></li>
<li><a href="links_nodes.jpg">links_nodes.jpg</a></li>
<li><a href="mininet/">mininet/</a></li>
<li><a href="tarea01.py">tarea01.py</a></li>
</ul>
<hr>
</body>
</html>
- 100%[=====>] 518 --.-KB/s en 0s
2024-03-30 21:17:03 (108 MB/s) - escritos a stdout [518/518]

```

Figura 15: Actividad 2

Se aprecia entonces que la llamada HTTP a este servidor h1 lo que mostrara es un documento HTML que contiene los archivos que hay en el directorio desde donde se ejecuto el script. Naturalmente, y como en cualquier llamada HTTP, existe un trafico vinculado a esta petición, este trafico fue capturado en wireshark, y es el siguiente:

6 2.560037440	fe80::d8c3:a5ff:fe4... ff02::2	ICMPv6	70 Router Solicitation from da:c3:a5:42:67:00
7 2.846112120	fe80::d00d:8bff:fe5... ff02::2	ICMPv6	70 Router Solicitation from d2:0d:8b:5a:84:44
8 3.057031237	fe80::300a:dfff:fe7... ff02::2	ICMPv6	70 Router Solicitation from 32:0a:df:7a:94:8e
9 7.447541131	3e:6f:ee:88:c5:89	Broadcast	ARP
10 7.447550459	82:67:9f:f2:c3:09	3e:6f:ee:88:c5:89	ARP
11 7.447712327	10.0.0.2	10.0.0.1	TCP
12 7.447735180	10.0.0.1	10.0.0.2	TCP
13 7.447851433	10.0.0.2	10.0.0.1	TCP
14 7.447915463	10.0.0.2	10.0.0.1	HTTP
15 7.447923094	10.0.0.1	10.0.0.2	TCP
16 7.448773596	10.0.0.1	10.0.0.2	TCP
17 7.448786967	10.0.0.2	10.0.0.1	TCP
18 7.448817964	10.0.0.1	10.0.0.2	HTTP
19 7.448823650	10.0.0.2	10.0.0.1	TCP
20 7.448858770	10.0.0.1	10.0.0.2	TCP
21 7.449345533	10.0.0.2	10.0.0.1	TCP
22 7.449359875	10.0.0.1	10.0.0.2	TCP
23 12.528918279	82:67:9f:f2:c3:09	3e:6f:ee:88:c5:89	ARP
24 12.529436327	3e:6f:ee:88:c5:89	82:67:9f:f2:c3:09	ARP
25 14.337129062	fe80::6493:f1ff:fe2... ff02::2	ICMPv6	70 Router Solicitation from 66:93:f1:21:1a:73
26 15.903254393	fe80::a072:fa:ff:fe1... ff02::2	ICMPv6	70 Router Solicitation from a2:72:fa:1f:03:6f
27 16.889908185	fe80::2c96:a1ff:fe7... ff02::2	ICMPv6	70 Router Solicitation from 2e:96:a1:70:16:78

Figura 16: Paquetes capturados actividad 2

Se puede ver en la imagen precedente como es que se establece una conexion y posteriormente se procede con la comunicacion entre las IP **10.0.0.1**, que corresponde al **h1**, y la ip **10.0.0.2** que corresponde a **h2**.

Dentro de este trafico podemos distinguir 2 protocolos importantes que no hemos descrito anteriormente:

1. **TCP:** El protocolo TCP es orientado a la conexión y garantiza la entrega confiable de paquetes en el orden correcto.
2. **HTTP:** El protocolo HTTP se utiliza para transferir archivos de texto, gráficos, sonido, vídeo y otros archivos multimedia desde un servidor web a un cliente web, en este caso, entre el servidor h1 y el cliente h2.

2.3. Actividad 3:

Ya avanzados en el mundo de mininet y sus funcionalidades y alcances, se va a testear un script que es parte de los ejemplos de la biblioteca. Este archivo **nat.py** va a crear una red con topología *tree*, o tipo árbol, que tendrá conexión a internet a través del **protocolo NAT**. El script diseñado por mininet es el siguiente:

```
from mininet.cli import CLI
from mininet.log import lg, info
from mininet.topolib import TreeNet

if __name__ == '__main__':
    lg.setLogLevel( 'info' )
    net = TreeNet( depth=1, fanout=4, waitConnected=True )
    # Add NAT connectivity
    net.addNAT().configDefault()
    net.start()
    info( "*** Hosts are running and should have internet\n" )
    info( "*** Type 'exit' or control-D to shut down network\n" )
    CLI( net )
    # Shut down NAT
    net.stop()
```

Al ejecutar este script con el clásico comando *sudo python3 nat.py*, se visualiza la siguiente respuesta del lado de mininet:

```

$ sudo python3 nat.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(s1, h1) (s1, h2) (s1, h3) (s1, h4)
*** Configuring hosts
h1 h2 h3 h4
*** Adding "iface nat0-eth0 inet manual" to /etc/network/interfaces
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Hosts are running and should have internet connectivity
*** Type 'exit' or control-D to shut down network
*** Starting CLI:
mininet>

```

Figura 17: Ejecución script nat.py

Se desprende de la imagen anterior que nuestra topología esta compuesta por un switch y 4 hosts conectados a el.

Para comprobar que efectivamente nuestra topología es capaz de conectarse a internet es que se va a pingear **www.google.com**; en caso de obtener respuesta positiva a nuestros paquetes ICMP entonces sabremos que la red fue configurada correctamente:

```

mininet> xterm h1
mininet> h1 ping -c 4 www.google.com
PING www.google.com (64.233.186.106) 56(84) bytes of data.
64 bytes from cb-in-f106.1e100.net (64.233.186.106): icmp_seq=1 ttl=58 time=7.95 ms
64 bytes from cb-in-f106.1e100.net (64.233.186.106): icmp_seq=2 ttl=58 time=9.49 ms
64 bytes from cb-in-f106.1e100.net (64.233.186.106): icmp_seq=3 ttl=58 time=8.33 ms
64 bytes from cb-in-f106.1e100.net (64.233.186.106): icmp_seq=4 ttl=58 time=16.1 ms

— www.google.com ping statistics —
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 7.947/10.476/16.146/3.321 ms
mininet>

```

Figura 18: pings a google

Se puede ver en la imagen anterior que los pings fueron enviados correctamente, y que también se recibió una respuesta satisfactoria de los mismos, esto se puede apreciar en la

ante penúltima línea, en donde se aprecia que hubieron 4 paquetes transmitidos, 4 recibidos y 0% de packet loss.

Al igual que en los casos anteriores, también en este caso en donde existe una conexión a internet es posible capturar los paquetes vía wireshark, esta captura se ilustra a continuación:

1	0.0.0.0.0.0.0.0.0.0	fe80::9c27:57ff:fe4::ff02::12	ICMPv6	70	Router Solicitation from 9e:27:57:4c:cd:c3
2	16.305942290	fe80::cc62:e6ff:fe2::ff02::12	ICMPv6	70	Router Solicitation from ce:62:e6:26:2c:ca
3	49.153092343	fe80::c8d4:d9ff:fe1::ff02::12	ICMPv6	70	Router Solicitation from ca:d4:d9:fe1:ad:93
4	66.202407936	9e:27:57:4c:cd:c3	Broadcast	ARP	42 Who has 10.0.0.5? Tell 10.0.0.1
5	66.203442666	aa:82:b8:58:64:d1	9e:27:57:4c:cd:c3	ARP	42 10.0.0.5 is at aa:82:b8:58:64:d1
6	66.203448085	10.0.0.1	200.28.4.130	DNS	74 Standard query 0x4f5e A www.google.com
7	66.203448474	10.0.0.1	200.28.4.130	DNS	74 Standard query 0x4f5e AAAA www.google.com
8	66.211366782	200.28.4.130	10.0.0.1	DNS	178 Standard query response 0x4f5e A www.google.com A 64.233.186.106 A 64.233.186.147 A 64.233.186.103
9	66.211414366	200.28.4.130	10.0.0.1	DNS	186 Standard query response 0x4f5e AAAA www.google.com AAAA 2800:3f0:4003:c00::93 AAAA 2800:3f0:4003:c
10	66.211671011	10.0.0.1	64.233.186.106	ICMP	98 Echo (ping) request id=0xea36, seq=1/256, ttl=64 (reply in 11)
11	66.219606140	64.233.186.106	10.0.0.1	ICMP	98 Echo (ping) reply id=0xea36, seq=1/256, ttl=58 (request in 10)
12	66.219879999	10.0.0.1	200.28.4.130	DNS	87 Standard query 0xea1e PTR 106.186.233.64.in-addr.arpa
13	66.227146027	200.28.4.130	10.0.0.1	DNS	121 Standard query response 0xea1e PTR 106.186.233.64.in-addr.arpa PTR cb-in-f106.1e100.net
14	67.213625847	10.0.0.1	64.233.186.106	ICMP	98 Echo (ping) request id=0xea36, seq=2/512, ttl=64 (reply in 15)
15	67.223067358	64.233.186.106	10.0.0.1	ICMP	98 Echo (ping) reply id=0xea36, seq=2/512, ttl=58 (request in 14)
16	67.223354791	10.0.0.1	200.28.4.130	DNS	87 Standard query 0x9c95 PTR 106.186.233.64.in-addr.arpa
17	67.231523392	200.28.4.130	10.0.0.1	DNS	121 Standard query response 0x9c95 PTR 106.186.233.64.in-addr.arpa PTR cb-in-f106.1e100.net
18	68.214819118	10.0.0.1	64.233.186.106	ICMP	98 Echo (ping) request id=0xea36, seq=3/768, ttl=64 (reply in 19)
19	68.223101067	64.233.186.106	10.0.0.1	ICMP	98 Echo (ping) reply id=0xea36, seq=3/768, ttl=58 (request in 18)
20	68.223361507	10.0.0.1	200.28.4.130	DNS	87 Standard query 0x58d7 PTR 106.186.233.64.in-addr.arpa
21	68.231601365	200.28.4.130	10.0.0.1	DNS	121 Standard query response 0x58d7 PTR 106.186.233.64.in-addr.arpa PTR cb-in-f106.1e100.net
22	69.210606674	10.0.0.1	64.233.186.106	ICMP	98 Echo (ping) request id=0xea36, seq=4/1024, ttl=64 (reply in 23)
23	69.232165606	64.233.186.106	10.0.0.1	ICMP	98 Echo (ping) reply id=0xea36, seq=4/1024, ttl=58 (request in 22)
24	69.232472523	10.0.0.1	200.28.4.130	DNS	87 Standard query 0x9926 PTR 106.186.233.64.in-addr.arpa
25	69.240619595	200.28.4.130	10.0.0.1	DNS	121 Standard query response 0x9926 PTR 106.186.233.64.in-addr.arpa PTR cb-in-f106.1e100.net
26	71.425218413	aa:82:b8:58:64:d1	9e:27:57:4c:cd:c3	ARP	42 Who has 10.0.0.1? Tell 10.0.0.5
27	71.425236795	9e:27:57:4c:cd:c3	aa:82:b8:58:64:d1	ARP	42 10.0.0.1 is at 9e:27:57:4c:cd:c3

Figura 19: Captura wireshark de pings a google

De la imagen anterior, además de ver los protocolos anteriormente descritos (ICMP, ARP, ICMPv6), que son los protocolos característicos involucrados en los pings, dado que ahora la comunicación es con una maquina externa, es decir, que esta ubicada virtualmente en la nube y físicamente en algún lugar del planeta, ahora aparece el **protocolo DNS**:

- **Protocolo DNS:** Cuando el host local (h1) intenta comunicarse con “www.google.com”, necesita saber la dirección IP correspondiente para establecer la conexión. Es en este punto que entra el protocolo DNS: El host envía una solicitud DNS para resolver el nombre del dominio “www.google.com” a su dirección IP, obteniendo las IP que podemos ver en la captura de wireshark.

Ya pinguado google.com, se solicita levantar un servidor en alguno de los nodos de nuestra topología tree. Para esto, se escoge el nodo h1 y se levanta el servidor en el, de la siguiente manera:

```
mininet> h1 python -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.0.0.5 - - [30/Mar/2024 23:02:49] "GET / HTTP/1.1" 200 -
10.0.0.5 - - [30/Mar/2024 23:02:49] code 404, message File not found
10.0.0.5 - - [30/Mar/2024 23:02:49] "GET /favicon.ico HTTP/1.1" 404 -
```

Figura 20: Levantamiento del servidor http en h1

Una vez levantado el servidor, es posible acceder a el mediante cualquier navegador de nuestra maquina, en este caso se utiliza **Firefox**, pues dado que estamos en nuestro sistema

Kali Linux nativo, no se cuenta con el navegador que viene por defecto en la VM de mininet. El servidor en cuestión comparte la siguiente información:

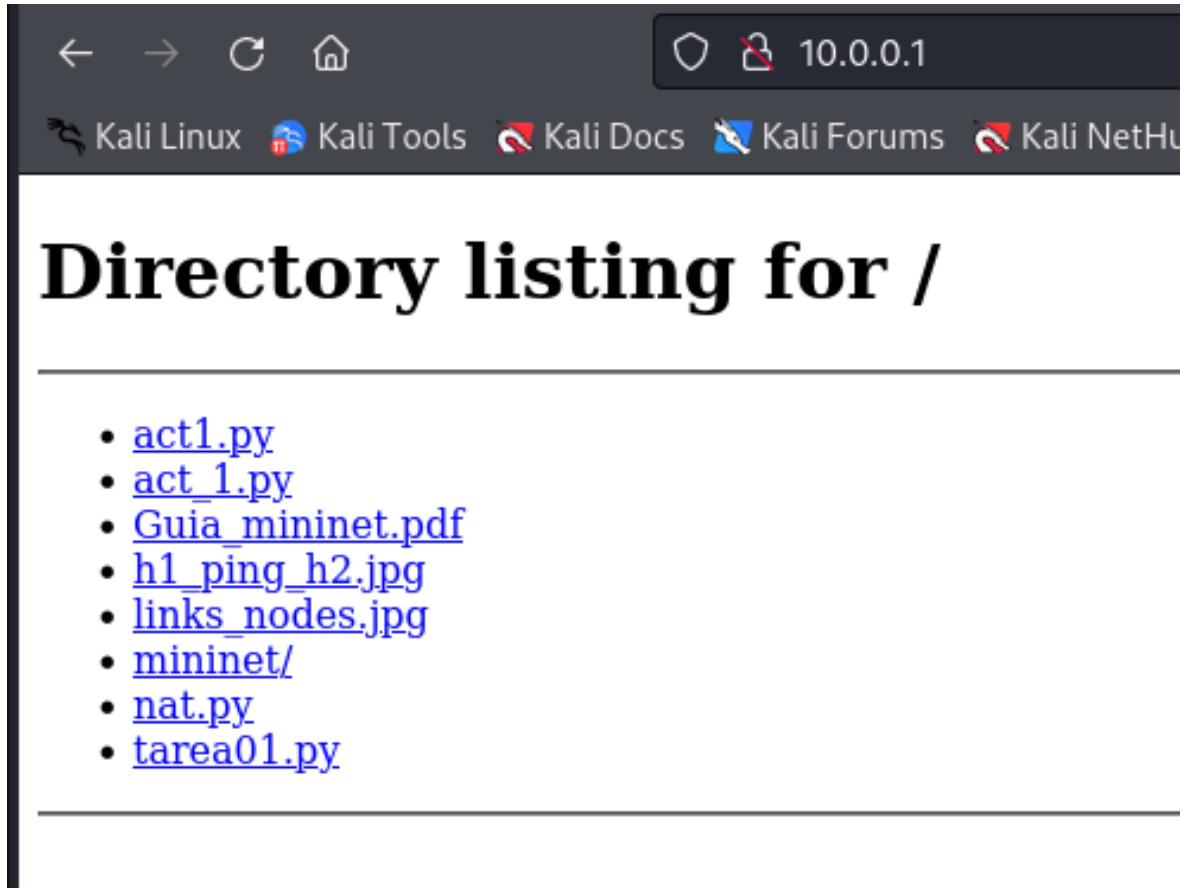


Figura 21: Servidor http, host1

Al igual que en los casos anteriores, también existe un tráfico generado asociado a esta petición http, a continuación:

26	71.425218413	aa:82:b8:58:64:d1	9e:27:57:4c:cd:c3	ARP	42	Who has 10.0.0.1? Tell 10.0.0.5
27	71.425236705	9e:27:57:4c:cd:c3	aa:82:b8:58:64:d1	ARP	42	10.0.0.1 is at 9e:27:57:4c:cd:c3
28	361.524742756	10.0.0.5	10.0.0.1	TCP	74	50756 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2216840195 TSecr=0 WS=512
29	361.524770274	10.0.0.1	10.0.0.5	TCP	74	80 → 50756 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSval=702230755 TSecr=
30	361.525117058	10.0.0.5	10.0.0.1	TCP	66	50750 → 80 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=2216840196 TSecr=702230755
31	361.525151339	10.0.0.5	10.0.0.1	HTTP	419	GET / HTTP/1.1
32	361.525161028	10.0.0.1	10.0.0.5	TCP	66	80 → 50756 [ACK] Seq=1 Ack=354 Win=43520 Len=0 TSval=702230755 TSecr=2216840196
33	361.526579435	10.0.0.1	10.0.0.5	TCP	221	80 → 50756 [PSH, ACK] Seq=1 Ack=354 Win=43520 Len=155 TSval=702230756 TSecr=2216840196 [TCP s
34	361.526628501	10.0.0.1	10.0.0.5	HTTP	621	HTTP/1.0 200 OK (text/html)
35	361.526794813	10.0.0.5	10.0.0.1	TCP	66	50756 → 80 [ACK] Seq=354 Ack=156 Win=42496 Len=0 TSval=2216840198 TSecr=702230756
36	361.526831557	10.0.0.5	10.0.0.1	TCP	66	50756 → 80 [ACK] Seq=354 Ack=712 Win=42496 Len=0 TSval=2216840198 TSecr=702230756
37	361.526944336	10.0.0.5	10.0.0.1	TCP	66	50756 → 80 [FIN, ACK] Seq=354 Ack=712 Win=42496 Len=0 TSval=2216840198 TSecr=702230756
38	361.526958259	10.0.0.1	10.0.0.5	TCP	66	80 → 50756 [ACK] Seq=712 Ack=355 Win=43520 Len=0 TSval=702230757 TSecr=2216840198
39	361.654907338	10.0.0.5	10.0.0.1	TCP	74	50750 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2216840325 TSecr=0 WS=512
40	361.654925981	10.0.0.1	10.0.0.5	TCP	74	80 → 50758 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SACK_PERM=1 TSval=702230885 TSecr=
41	361.655241472	10.0.0.5	10.0.0.1	TCP	66	50758 → 80 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=2216840326 TSecr=702230885
42	361.655276074	10.0.0.5	10.0.0.1	HTTP	367	GET /favicon.ico HTTP/1.1
43	361.655285029	10.0.0.1	10.0.0.5	TCP	66	80 → 50758 [ACK] Seq=1 Ack=302 Win=43520 Len=0 TSval=702230885 TSecr=2216840326
44	361.659705955	10.0.0.1	10.0.0.5	TCP	251	80 → 50758 [PSH, ACK] Seq=1 Ack=302 Win=43520 Len=185 TSval=702230889 TSecr=2216840326 [TCP s
45	361.659776637	10.0.0.1	10.0.0.5	HTTP	401	HTTP/1.0 404 File not found (text/html)
46	361.660125994	10.0.0.5	10.0.0.1	TCP	66	50750 → 80 [ACK] Seq=302 Ack=186 Win=42496 Len=0 TSval=2216840331 TSecr=702230889
47	361.660242931	10.0.0.5	10.0.0.1	TCP	66	50758 → 80 [FIN, ACK] Seq=302 Ack=522 Win=42496 Len=0 TSval=2216840331 TSecr=702230890
48	361.660255870	10.0.0.1	10.0.0.5	TCP	66	80 → 50758 [ACK] Seq=522 Ack=303 Win=43520 Len=0 TSval=702230890 TSecr=2216840331
49	409.601223864	fe80::3051:43ff:fed_	ff02::2	ICMPv6	70	Router Solicitation from 32:51:43:df:97:d6
50	442.367764333	fe80::2c31:faff:fe5_	ff02::2	ICMPv6	70	Router Solicitation from 2e:31:fa:59:86:f8
51	458.753112800	fe80::a882:b8ff:fe5_	ff02::2	ICMPv6	70	Router Solicitation from aa:82:b8:58:64:d1
52	491.519768894	fe80::9c27:57ff:fe4_	ff02::2	ICMPv6	70	Router Solicitation from 9e:27:57:4c:cd:c3

Figura 22: Captura wireshark servidor h1

De esta captura se puede ver un trafico clásico para este tipo de interacciones, con un servidor, utilizando los mismos protocolos que hemos descrito anteriormente.

2.4. Actividad 4:

Finalmente, y ya en la ultima actividad, se solicita diseñar una topología personalizada con 2 hosts (hostChile y hostAustralia) y 4 switches entre ellos. La conexión entre los switches también es personalizada, teniendo valores de BW, delay y loss distinto para cada link. Una vez diseñada y levantada la red, se solicita hacer una transferencia de archivos vía FTP, para esto se configurara el hostAustralia (en la topología sera h2) como el servidor FTP. De este modo, la red a diseñar es la siguiente:

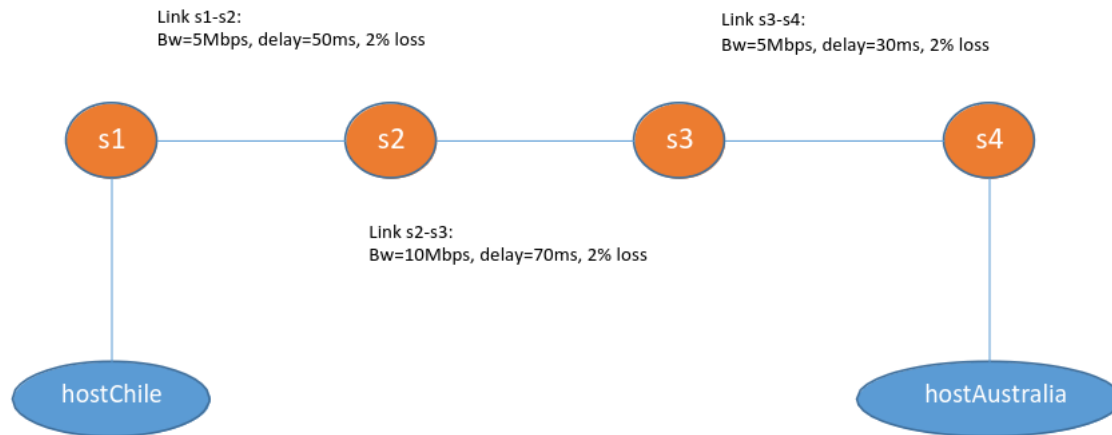


Figura 23: Topología de red, actividad 4

Importante: Los primeros 3 pasos de la presente actividad fueron realizados según las descripciones del equipo especificadas en la introducción del documento, es decir, kali Linux nativo. Sin embargo, esta metodología presento muchos problemas para realizar el punto 4, pues al no trabajar con la imagen de mininet que se recomienda, fueron necesarias muchas instalaciones y configuraciones que en su conjunto impidieron el correcto desarrollo del punto 4 de esta actividad. Es por esto que para hacer el envío del archivo vía FTP se decidió instalar la maquina virtual según los pasos indicados en la descripción de la actividad.

Para cumplir con la topología solicitada, el código diseñado es el siguiente:

```

from mininet.topo import Topo

class Act_4(Topo) :
    " Topologia Simple"
  
```

```

def build(self) :
    # hosts
    h_Chile = self.addHost('h1')
    h_Australia = self.addHost('h2')

    # A adiando Switchs
    s1 = self.addSwitch('s1')
    s2 = self.addSwitch('s2')
    s3 = self.addSwitch('s3')
    s4 = self.addSwitch('s4')

    # Add links
    self.addLink( h_Chile , s1 )
    self.addLink( s1 , s2 , bw =5, delay="50ms", loss =2)
    self.addLink( s2 , s3 , bw =10, delay="70ms", loss =2)
    self.addLink( s3 , s4 , bw =5, delay="30ms", loss =2)
    self.addLink( h_Australia , s4 )

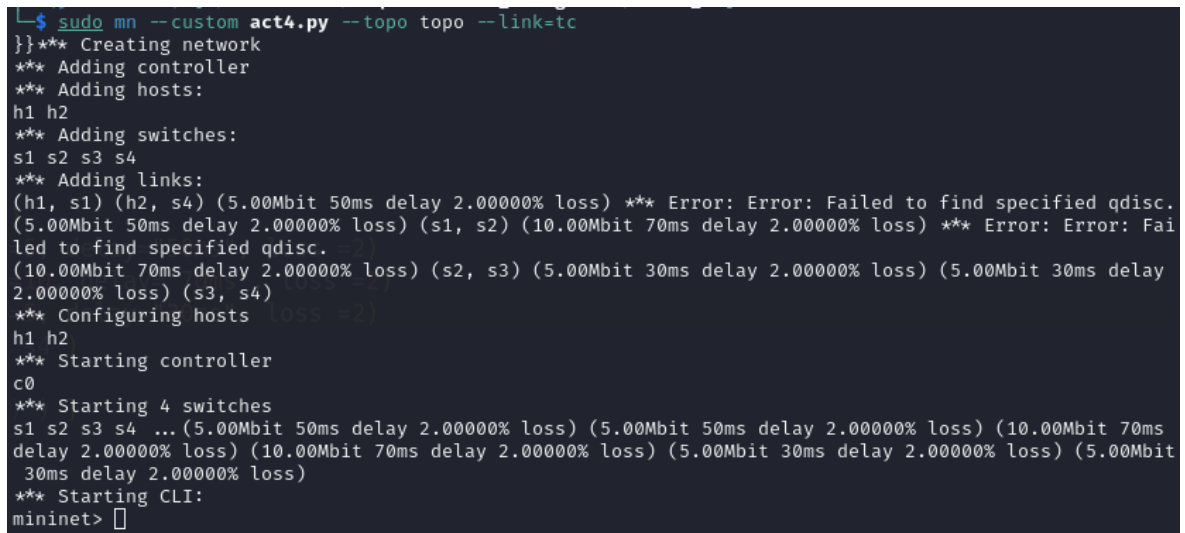
topos = {"topo": ( lambda : Act_4 () ) }

```

Bien, ya definida la topología y el código que va a hacer posible su construcción, se procede a ejecutar este script vía el siguiente comando:

```
sudo mn --custom act4.py --topo topo --link=tc
```

Al ejecutar el script con el comando anterior, se obtiene el siguiente resultado:



```

$ sudo mn --custom act4.py --topo topo --link=tc
}}*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s4) (5.00Mbit 50ms delay 2.00000% loss) *** Error: Error: Failed to find specified qdisc.
(5.00Mbit 50ms delay 2.00000% loss) (s1, s2) (10.00Mbit 70ms delay 2.00000% loss) *** Error: Error: Fai
led to find specified qdisc.
(10.00Mbit 70ms delay 2.00000% loss) (s2, s3) (5.00Mbit 30ms delay 2.00000% loss) (5.00Mbit 30ms delay
2.00000% loss) (s3, s4)
*** Configuring hosts loss =2)
h1 h2
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ... (5.00Mbit 50ms delay 2.00000% loss) (5.00Mbit 50ms delay 2.00000% loss) (10.00Mbit 70ms
delay 2.00000% loss) (10.00Mbit 70ms delay 2.00000% loss) (5.00Mbit 30ms delay 2.00000% loss) (5.00Mbit
30ms delay 2.00000% loss)
*** Starting CLI:
mininet>

```

Figura 24: Levantando red, actividad 4

Ahora ya tenemos lo necesario para seguir con las siguientes pasos de esta ultima actividad, estos estan detallados a continuacion:

2.4.2. Paso II: Transferencia de archivo usando FTP

Ya testeada la red (en el apartado anterior), se seguirá una serie de pasos para lograr enviar un archivo vía FTP desde h2 a h1. Estos pasos detallados a continuación:

1. Generar un archivo de 10 MB

Para esto, se hace uso del siguiente comando:

```
fallocate -l 10MiB un-archivo.png
```

Este comando va a generar un archivo con el tamaño y nombre especificado, que posteriormente sera enviado vía FTP entre los host Chile y Australia. Esto se ve de la siguiente manera:

```
mininet@mininet-vm:~/mininet/custom$ fallocate -l 10MiB archivo.png
mininet@mininet-vm:~/mininet/custom$ ls
act4.py  archivo.png  h1.log  h2.log  README  topo-2sw-2host.py
```

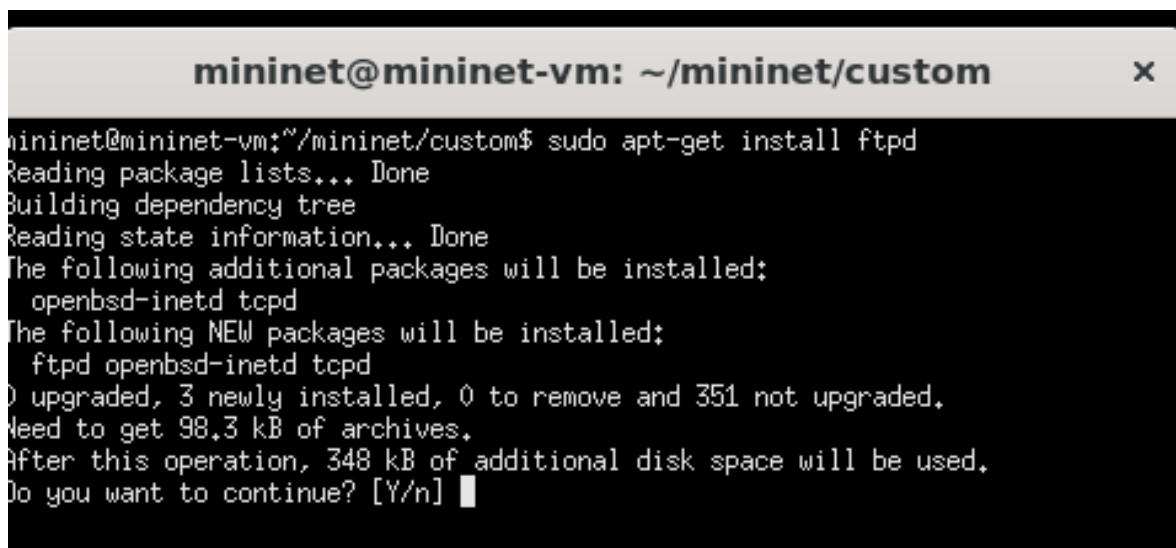
Figura 27: Comando fallocate

2. Instalar un servidor FTP

A continuacion, se debe instalar un servido FTP tanto para la conexión como para la descarga del archivo. Para hacer esto, se sigue la siguiente instrucción:

```
sudo apt-get install ftpd
```

Resultado de esta situación obtenemos lo siguiente:



```
mininet@mininet-vm: ~/mininet/custom x
mininet@mininet-vm:~/mininet/custom$ sudo apt-get install ftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  openbsd-inetd tcpd
The following NEW packages will be installed:
  ftpd openbsd-inetd tcpd
0 upgraded, 3 newly installed, 0 to remove and 351 not upgraded.
Need to get 98,3 kB of archives.
After this operation, 348 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Figura 28: Instalación servidor FTP

3. Iniciar terminales simultáneamente en h1 y h2

Una vez instalado el servidor FTP se deben iniciar simultáneamente 2 terminales con `xterm`, una para h1 y otra para h2. Para lograr este cometido, se utiliza el comando: `xterm h1 h2`. Resultado de esta instrucción, se despliega 1 terminal para cada nodo, de la siguiente manera:

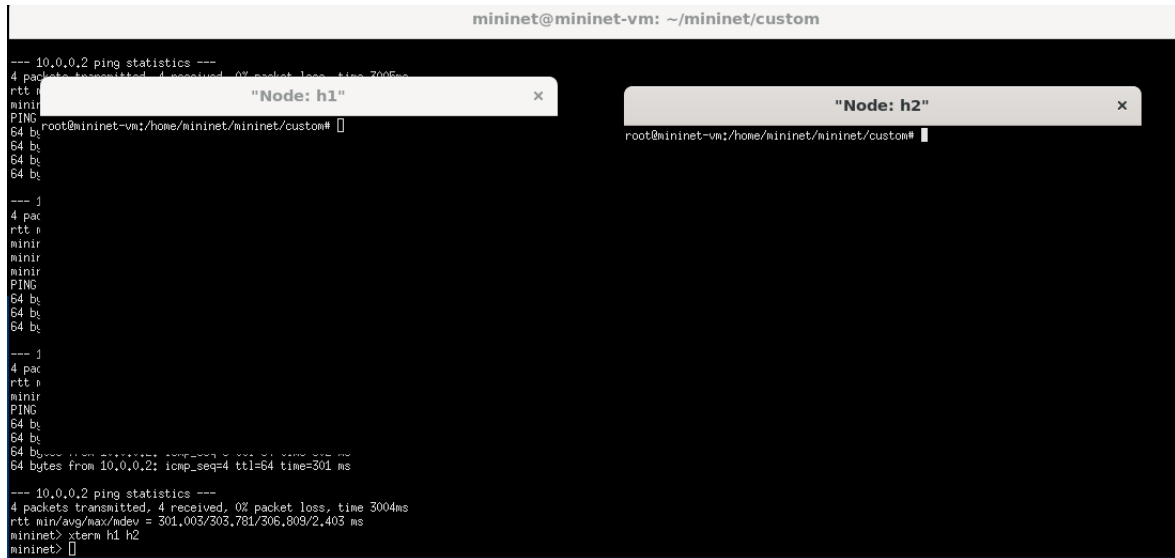


Figura 29: xterm h1 h2

4. Iniciar captura tcpdump

Se inicia de este modo una captura `.tcpdump` para paquetes entre h1 y h2. De la siguiente manera:

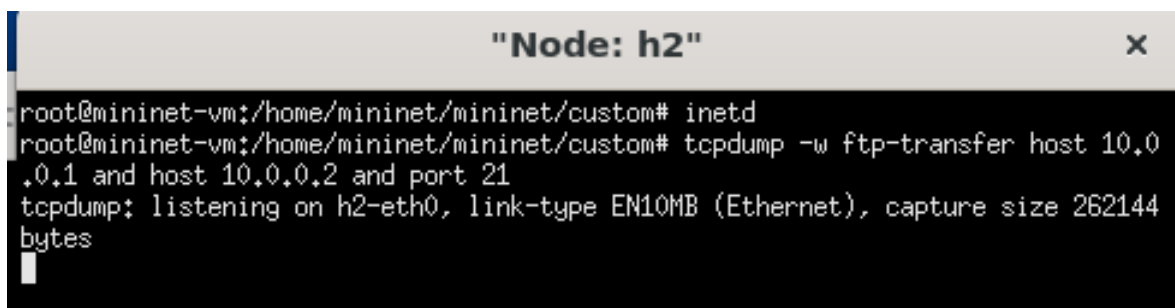
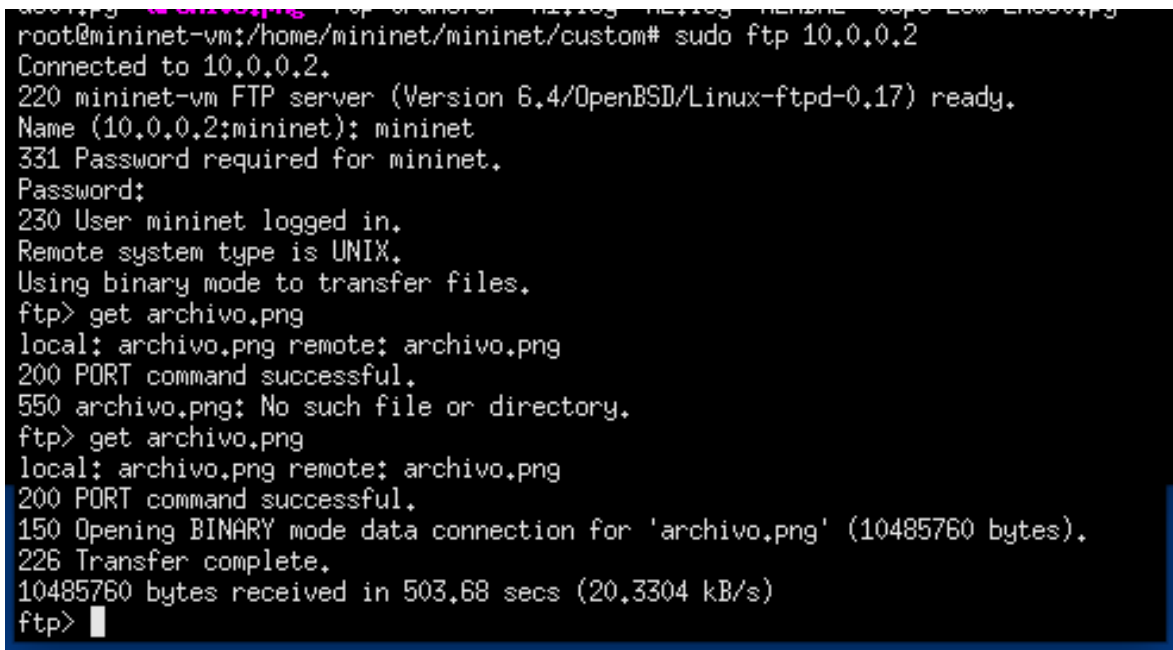


Figura 30: Captura tcpdump

5. Iniciar un servidor FTP e iniciar sesión en host 2 y obtención del archivo creado

Habiendo seguido todos los pasos anteriores, en este punto será fácil iniciar el servidor FTP y la sesión en el host 2, para esto, se debe hacer uso en primera instancia del comando `inetd` en la terminal del h2, posteriormente iniciar la sesión con la instrucción: `sudo ftp 10.0.0.2`. Finalmente, y una vez iniciada la sesión FTP en el host 2, se debe descargar el archivo creado en pasos anteriores. Para hacer esto se hace uso del comando `get archivo.png`, dando, de esta manera, por terminada la actividad. La situación recientemente descrita se ve de la siguiente manera:



```
root@mininet-vm:/home/mininet/mininet/custom# sudo ftp 10.0.0.2
Connected to 10.0.0.2.
220 mininet-vm FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (10.0.0.2:mininet): mininet
331 Password required for mininet.
Password:
230 User mininet logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get archivo.png
local: archivo.png remote: archivo.png
200 PORT command successful.
550 archivo.png: No such file or directory.
ftp> get archivo.png
local: archivo.png remote: archivo.png
200 PORT command successful.
150 Opening BINARY mode data connection for 'archivo.png' (10485760 bytes).
226 Transfer complete.
10485760 bytes received in 503.68 secs (20.3304 kB/s)
ftp> █
```

Figura 31: Conexión FTP y descarga del archivo

Se adjunta también evidencia de la captura FTP de la transmisión del archivo generado:

No.	Time	Source	Destination	Protocol	Length	Info
1441.	383.778770966	127.0.0.1	127.0.0.1	TCP	68	52978 → 6653 [ACK] Seq=6042614 Ack=6571698 Win=2427 Len=0 TSv...
1441.	383.781065742	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [PSH, ACK] Seq=5312713 Ack=1 Wi...
1441.	383.781067213	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [PSH, ACK] Seq=5312713 Ack=1 Wi...
1441.	383.781126947	10.0.0.2	10.0.0.1	OpenFL...	1600	Type: OFPT PACKET IN
1441.	383.781144591	127.0.0.1	127.0.0.1	OpenFL...	148	Type: OFPT FLOW MOD
1441.	383.781149578	10.0.0.2	10.0.0.1	OpenFL...	1606	Type: OFPT PACKET OUT
1441.	383.781161125	127.0.0.1	127.0.0.1	TCP	68	52978 → 6653 [ACK] Seq=6044146 Ack=6573316 Win=2427 Len=0 TSv...
1441.	383.800088823	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [ACK] Seq=5314161 Ack=1 Win=424...
1441.	383.800092036	10.0.0.2	10.0.0.1	TCP	1516	[TCP Retransmission] 20 → 40285 [PSH, ACK] Seq=5315609 Ack=1 ...
1441.	383.800091108	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [ACK] Seq=5314161 Ack=1 Win=424...
1441.	383.800092381	10.0.0.2	10.0.0.1	TCP	1516	[TCP Retransmission] 20 → 40285 [PSH, ACK] Seq=5315609 Ack=1 ...
1441.	383.800185405	10.0.0.2	10.0.0.1	OpenFL...	1600	Type: OFPT PACKET IN
1441.	383.800209857	127.0.0.1	127.0.0.1	OpenFL...	148	Type: OFPT FLOW MOD
1441.	383.800215392	10.0.0.2	10.0.0.1	OpenFL...	1606	Type: OFPT PACKET OUT
1441.	383.800226877	10.0.0.2	10.0.0.1	OpenFL...	1600	Type: OFPT PACKET IN
1441.	383.800238272	127.0.0.1	127.0.0.1	OpenFL...	148	Type: OFPT FLOW MOD
1441.	383.800242504	10.0.0.2	10.0.0.1	OpenFL...	1606	Type: OFPT PACKET OUT
1441.	383.800251771	127.0.0.1	127.0.0.1	TCP	68	52978 → 6653 [ACK] Seq=6047210 Ack=6576552 Win=2426 Len=0 TSv...
1442.	383.807007914	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [ACK] Seq=5306921 Ack=1 Win=424...
1442.	383.807009487	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [ACK] Seq=5306921 Ack=1 Win=424...
1442.	383.807076211	10.0.0.2	10.0.0.1	OpenFL...	1600	Type: OFPT PACKET IN
1442.	383.807095083	127.0.0.1	127.0.0.1	OpenFL...	148	Type: OFPT FLOW MOD
1442.	383.807100283	10.0.0.2	10.0.0.1	OpenFL...	1606	Type: OFPT PACKET OUT
1442.	383.807110974	127.0.0.1	127.0.0.1	TCP	68	52980 → 6653 [ACK] Seq=5934870 Ack=6461718 Win=83 Len=0 TSval...
1442.	383.846659979	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [ACK] Seq=5308369 Ack=1 Win=424...
1442.	383.846664600	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [PSH, ACK] Seq=5309817 Ack=1 Wi...
1442.	383.846663648	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [ACK] Seq=5308369 Ack=1 Win=424...
1442.	383.846665032	10.0.0.2	10.0.0.1	TCP	1516	[TCP Out-Of-Order] 20 → 40285 [PSH, ACK] Seq=5309817 Ack=1 Wi...
1442.	383.846845761	10.0.0.2	10.0.0.1	OpenFL...	1600	Type: OFPT PACKET IN
1442.	383.846880575	127.0.0.1	127.0.0.1	OpenFL...	148	Type: OFPT FLOW MOD
1442.	383.846887134	10.0.0.2	10.0.0.1	OpenFL...	1606	Type: OFPT PACKET OUT
1442.	383.846899586	10.0.0.2	10.0.0.1	OpenFL...	1600	Type: OFPT PACKET IN

Figura 32: Captura wireshark transferencia FTP

3. Análisis de resultados

Luego de haber realizado con éxito cada una de las actividades propuestas, y conseguido los distintos objetivos, es que se puede dar respuesta a preguntas como *¿Que tipo de problemas estan presentes en una comunicación TCP de paquetes?* Y es que el protocolo TCP, por definición, es un protocolo que confirma la correcta recepción de los paquetes enviados desde el transmisor, es por esta dinámica que pueden ocurrir diversos errores en la comunicación. A continuacion, se detallaran los errores que podemos encontrar durante el tiempo que estuvo activo este enlace.

1. **Out-of-Order (Fuera de orden):** Estos errores indican que los paquetes no están llegando en el orden en que fueron enviados. Puede ser causado por la pérdida de paquetes o retrasos en la red.
2. **Spurious Retransmission (Retransmisión espuria):** Estos errores ocurren cuando un paquete se retransmite innecesariamente. Puede deberse a pérdida de paquetes o congestión en la red.
3. **TCP Zero Window (Ventana TCP en cero):** Este error indica que el receptor no tiene espacio en su búfer para recibir más datos.
4. **TCP Dup ACK (ACK duplicado):** Estos ACK duplicados pueden ser causados por paquetes perdidos o fuera de orden.
5. **TCP Retransmission (Retransmisión TCP):** Ocurre cuando un paquete se reenvía debido a la falta de confirmación del receptor.

En general, para poder mitigar o prevenir este tipo de errores asociados a la comunicación TCP se pueden considerar los siguientes aspectos:

1. Asegurar de que la red esté bien configurada y optimizada.
2. Considerar aumentar la capacidad de la red si es necesario para manejar el tráfico.
3. Utilizar herramientas y técnicas para monitorear y gestionar el rendimiento de la red.
4. Verificar la calidad de la conexión entre los hosts.
5. Aumentar el tamaño del búfer en el receptor.
6. Optimizar la configuración de la ventana TCP.
7. Considerar implementar mecanismos de retransmisión selectiva, como por ejemplo el SACK.[1]

Es importante mencionar que esta gran cantidad de errores, en gran medida viene dado porque la red fue configurada para tener un determinado porcentaje de error solicitado en el enunciado de la actividad 4.

Ahora bien, ya identificamos los errores asociados a este tipo de enlaces y comunicaciones, y pudimos observar como la gran mayoría de los errores TCP presentes en esta transferencia de información están asociados a la confirmación de la recepción de los paquetes enviados. Entonces la pregunta que surge lógicamente sería: ***¿Sería conveniente usar un enlace UDP para la transferencia de archivos en un enlace de este tipo?*** . Y es que la respuesta en este caso, como es habitual en el mundo de la informática es: **depende**. Esto se debe a que la comunicación UDP es una comunicación que se basa en la transmisión de información sin una confirmación de correcta recepción asociada a cada paquete. Dada esta característica fundamental, es que la transmisión de paquetes, naturalmente, es mas rápida y fluida; sin embargo, esta ventaja tiene un contra asociado, y es que dado que no hay ninguna confirmación sobre una correcta recepción del paquete, el receptor no sabe si se perdió información, y de esta forma continua enviando paquetes sin parar hasta que termine de enviar todo. En casos como streaming, o televisión digital, por ejemplo, la comunicación UDP es ideal, pues es un flujo constante de información, y en caso de perderse algún(os) paquetes no sería una situación tan terrible, y no tendría un gran impacto en la calidad de la imagen o sonido (salvo que sean muchos paquetes, en ese caso se vería afectada la señal). Para el caso de archivos, por ejemplo, una imagen como lo es este caso, sería mas conveniente utilizar el protocolo TCP, pues si perdemos paquetes podríamos perder información importante para la imagen, lo que hará que no la veamos bien. En el caso de un archivo word, por ejemplo, o un archivo PDF, también es de vital importancia tener toda la información, pues en caso contrario podríamos corromper el archivo.

4. Conclusiones y comentarios

Tal como vimos en el presente documento, y luego de la realización de diversas actividades relacionadas con la construcción de redes virtuales a través de mininet, es que se pueden sacar varias conclusiones al respecto.

En primer lugar, destacar la importancia de hacer un análisis previo a la hora de diseñar distintos tipos de redes, analizar su instalación, configuración, dificultades asociadas y factores a considerar, como delay, BW, ventana TCP, etc. Pues cada uno de estos parametros es fundamental a la hora de un correcto funcionamiento de la red, tanto para evitar la congestión, evitar retransmisiones de paquetes innecesarias y así lograr una comunicación eficiente.

Además de esto, se espera también que el lector haya podido identificar claramente los errores asociados a una comunicación TCP en esta red virtual, dando una especial consideración a los parametros configurados (BW, Loss, delay), pues estos son factores que afectaran la comunicación. Además, al analizar estos errores, se desprenden también medidas a tomar para evitar este tipo de problemas, o al menos mitigarlos.

Finalmente, se confirma que la alumna consiguió los objetivos planteados al principio de este informe, comprender el funcionamiento de una red virtual en mininet, considerando factores asociados e inherentes a la comunicación entre distintos host, analizando los problemas que pueden haber en una comunicación y transmisión de datos vía FTP (TCP), versus las ventajas y desventajas que trae hacer esta misma transición pero utilizando el protocolo UDP.

https://github.com/iJass21/Arquitecturas_Emergentes

Referencias

- [1] Barreto, Migue; Belino, Maximiliano; San Martín, Marcelo, “TCP SACK”; https://d1wqtxts1xzle7.cloudfront.net/34159420/TCPsack-libre.pdf?1404930347=&response-content-disposition=inline%3B+filename%3DTCP_SACK_Selective_Acknowledge_o_Reconoc.pdf&Expires=1712440411&Signature=gpszBbIbvAQLzkV2lk6udrx6YMko2bXy60JXfICsb-9SjtzgrSV1Q1Ra3msIvUUYV5gMEy9sz07Flx0r33c0-&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA