

Tarea 1 Arquitecturas Emergentes

Sección 1

Pablo Díaz Chamorro
e-mail: pablo.diaz_c@mail.udp.cl

Septiembre de 2023

Índice

1. Introducción	2
2. Actividades	2
2.1. Actividad 1	2
2.1.1. Código	2
2.1.2. Implementación	3
2.1.3. Direcciones IPs hosts	3
2.1.4. 10 Pings	4
2.1.5. N pings con error del 5 %	7
2.2. Actividad 2	7
2.3. Actividad 3	8
2.3.1. Ejecución script nat.py y error nameserver	8
2.3.2. Solución error nameserver	9
2.3.3. Pruebas ping desde hosts hacia un DNS	11
2.3.4. Ejecución de servidor HTTP en host 2 e ingreso pagina web desde host 3	15
2.4. Actividad 4	17
2.4.1. Iperf entre host Chile y host australia	18
2.4.2. Transferencia de archivos usando FTP	18
2.4.3. Captura y análisis de trafico en Wireshark	21
3. GitHub	24

1. Introducción

El presente informe se focaliza en la ejecución de la tarea 1 del curso de arquitecturas emergentes, la cual tiene como objetivo familiarizar con la plataforma de emulación de redes, Mininet, y ejecutar distintas practicas. También, se aborda el concepto de Software Defined Networks(SDN), un paradigma de redes que integra recursos programables para reducir la dependencia del hardware usual en redes de computadores. Durante el desarrollo de esta tarea, se abordaran comandos básicos de Mininet, la instalación de Mininet y se simulara una tele-operación a larga distancia. También se introduce a la API de Python con el objetivo de diseñar topologías personalizadas y se usan herramientas como Wireshark y TCPDUMP, para el análisis del trafico de red y transferencia de archivos de las actividades correspondientes.

2. Actividades

Primeramente, se instala e inicia la maquina virtual a través de la imagen de Mininet en Virtual Box, siguiendo los pasos dichos en la tarea. Luego, se procede a realizar cada una de las actividades correspondientes:

2.1. Actividad 1

2.1.1. Código

En esta actividad se implementa una red en Mininet a través de la API de Python en el siguiente código:

```

from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        Host1 = self.addHost( 'h1' )
        Host2 = self.addHost( 'h2' )
        Host3 = self.addHost( 'h3' )
        Host4 = self.addHost( 'h4' )

        Switch1 = self.addSwitch( 's1' )
        Switch2 = self.addSwitch( 's2' )
        Switch3 = self.addSwitch( 's3' )
        Switch4 = self.addSwitch( 's4' )

        # Add links
        self.addLink( Host1, Switch1 )
        self.addLink( Host2, Switch1 )
        self.addLink( Switch1, Switch2 )
        self.addLink( Switch2, Switch3, bw=5, delay='20ms', loss=10)# especial
        self.addLink( Host3, Switch3 )
        self.addLink( Switch3, Switch4, bw=15, delay='40ms', loss=2 )# Especial
        self.addLink( Host4, Switch4 )

topos = { 'mytopo': ( lambda: MyTopo() ) }

```

Figura 1: Código Python actividad 1.

2.1.2. Implementación

Para poder implementar la topología creada a partir del código anterior en la API de Python, se ejecuta de esta forma:

```

mininet@mininet-vm:~/mininet/custom$ sudo mn --custom test1.py --topo mytopo --link=tc
*** 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) (5.00Mbit 20ms delay 10.00000% loss) (s2, s3) (15.00Mbit 40ms delay 2.00000% loss) (15.00Mbit 40ms delay 2.00000% loss) (s3, s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ... (5.00Mbit 20ms delay 10.00000% loss) (5.00Mbit 20ms delay 10.00000% loss) (15.00Mbit 40ms delay 2.00000% loss) (15.00Mbit 40ms delay 2.00000% loss)
*** Starting CLI:

```

Figura 2: Ejecución comando Python actividad 1.

2.1.3. Direcciones IPs hosts

Luego de haber ejecutado la topología creada, se procede a mostrar las direcciones IPs de cada uno de los host creados en la topología usando **ifconfig**.

```

mininet> h1 ifconfig -a
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
    ether 06:59:8a:15:2f:70 txqueuelen 1000 (Ethernet)
    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

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 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

mininet>

```

Figura 3: Dirección IP host 1 con ifconfig.

```

mininet> h2 ifconfig -a
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
    ether 06:59:8a:15:2f:65 txqueuelen 1000 (Ethernet)
    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

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 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

mininet>

```

Figura 4: Dirección IP host 2 con ifconfig.

```

mininet> h3 ifconfig -a
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
    ether d2:00:36:42:e5:5a txqueuelen 1000 (Ethernet)
    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

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 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

mininet>

```

Figura 5: Dirección IP host 3 con ifconfig.

```

mininet> h4 ifconfig -a
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
    ether 6a:1b:d3:4d:6a:f2 txqueuelen 1000 (Ethernet)
    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

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    loop txqueuelen 1000 (Local Loopback)
    RX packets 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

mininet>

```

Figura 6: Dirección IP host 4 con ifconfig.

2.1.4. 10 Pings

En primer lugar, se procede a ejecutar 10 paquetes ping que serán capturados a través de Wireshark, desde el host 1 hacia el host 2:

```

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=3.66 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.593 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.187 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.177 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.120 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.086 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.126 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.115 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.128 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9180ms
rtt min/avg/max/mdev = 0.086/0.530/3.663/1.053 ms

```

Figura 7: 10 paquetes ping desde host 1 a host 2

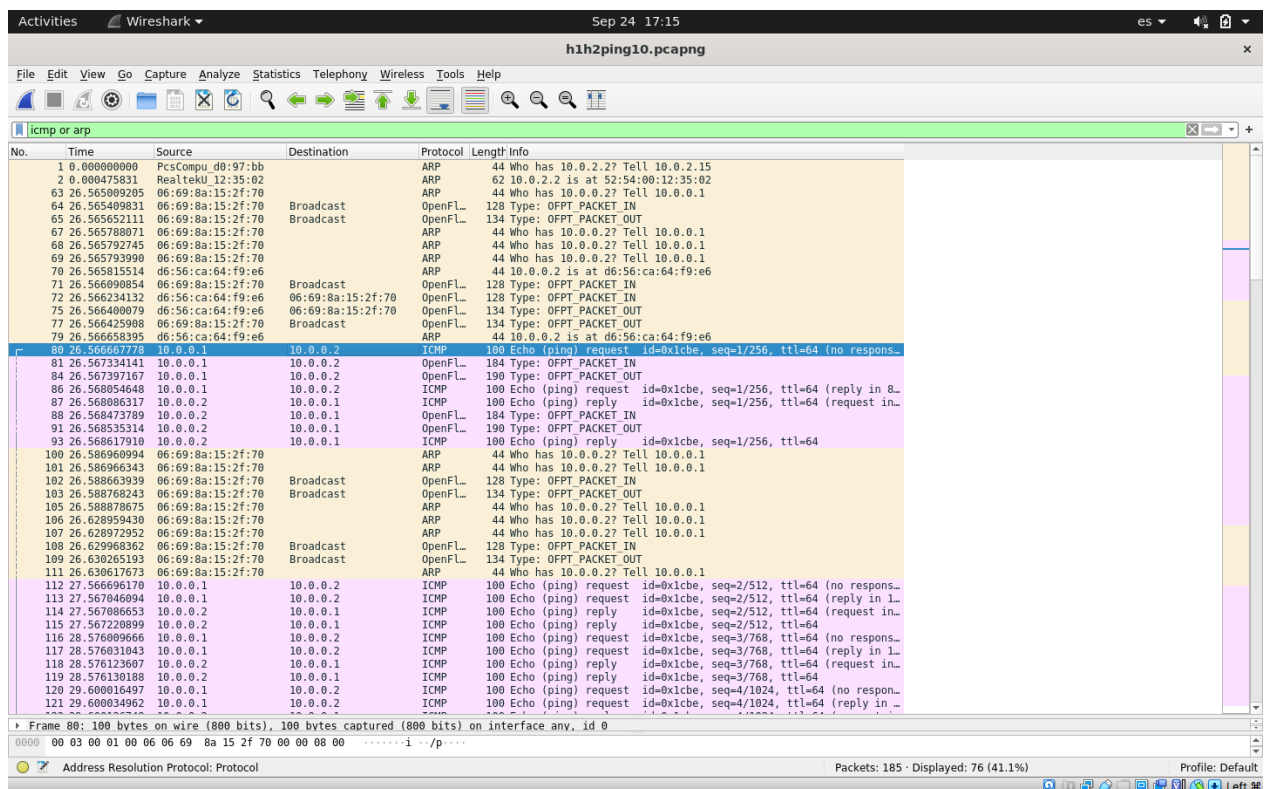


Figura 8: Captura 1 pings Wireshark con filtro icmp or arp

En segundo lugar, se repite la ejecución de 10 paquetes ping que serán capturados a través de Wireshark, desde el host 1 hacia el host 3:

```

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=97.6 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=41.8 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=40.6 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=40.4 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=41.3 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=40.4 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=40.7 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=40.3 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=41.3 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=41.1 ms

--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9023ms
rtt min/avg/max/mdev = 40.344/46.547/97.559/17.010 ms
mininet>

```

Figura 9: 10 paquetes ping desde host 1 a host 3

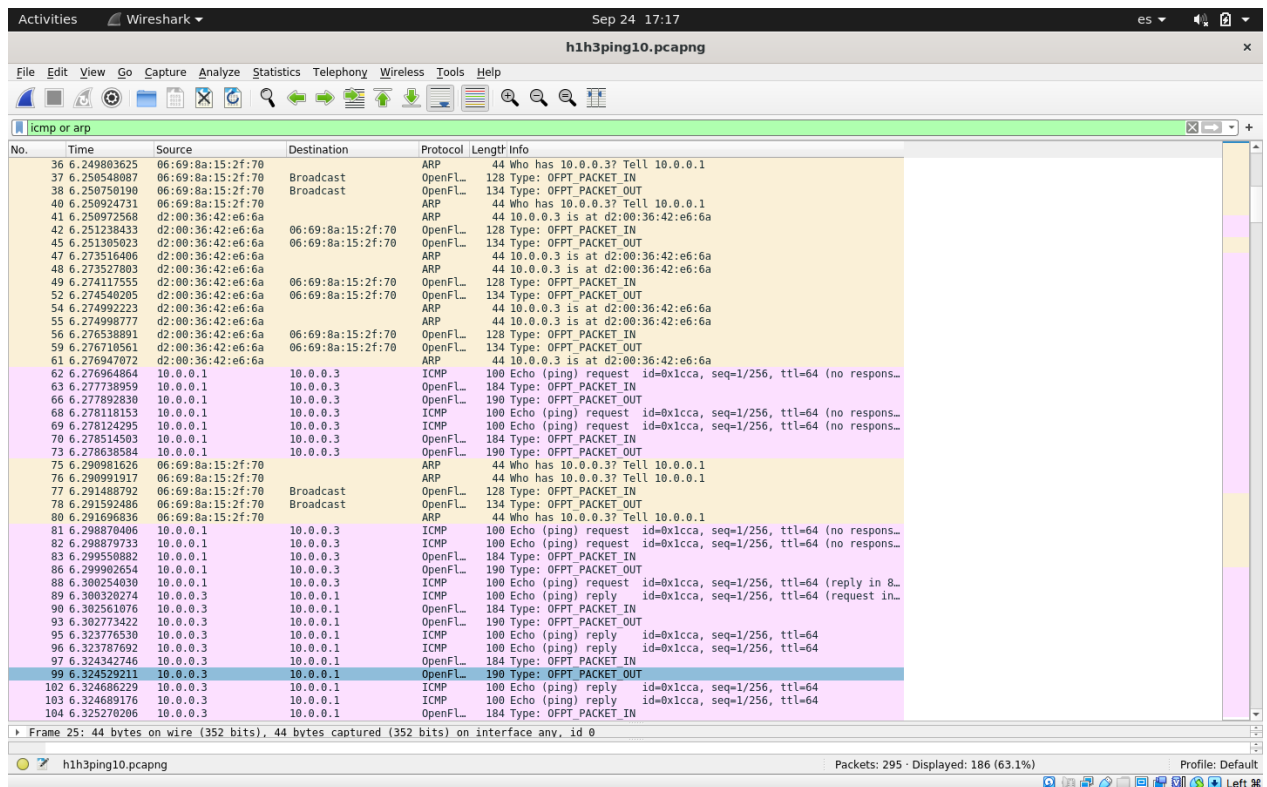


Figura 10: Captura 2 pings Wireshark con filtro icmp or arp

En las imágenes anteriores se puede observar como existe un retardo asociado a los enlaces del host 1 con el host 2, esto es debido a las configuraciones hechas en la topología de red, en el link entre el switch 3 y switch 4. Estas implicaciones son debido al parámetro puesto con delay igual a 40ms.

2.1.5. N pings con error del 5 %

En este caso se ejecutan una cantidad de N paquetes entre los host 3 y host 4, tal que se logre un error del 5 % en el porcentaje de perdida.

```
mininet> h3 ping -c 40 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=1176 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=174 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=81.9 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=81.5 ms
64 bytes from 10.0.0.4: icmp_seq=6 ttl=64 time=81.2 ms
64 bytes from 10.0.0.4: icmp_seq=7 ttl=64 time=81.1 ms
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=80.4 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=80.6 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=80.4 ms
64 bytes from 10.0.0.4: icmp_seq=11 ttl=64 time=80.8 ms
64 bytes from 10.0.0.4: icmp_seq=12 ttl=64 time=80.5 ms
64 bytes from 10.0.0.4: icmp_seq=14 ttl=64 time=81.5 ms
64 bytes from 10.0.0.4: icmp_seq=15 ttl=64 time=81.0 ms
64 bytes from 10.0.0.4: icmp_seq=16 ttl=64 time=80.9 ms
64 bytes from 10.0.0.4: icmp_seq=17 ttl=64 time=80.8 ms
64 bytes from 10.0.0.4: icmp_seq=18 ttl=64 time=81.1 ms
64 bytes from 10.0.0.4: icmp_seq=19 ttl=64 time=80.4 ms
64 bytes from 10.0.0.4: icmp_seq=20 ttl=64 time=81.0 ms
64 bytes from 10.0.0.4: icmp_seq=21 ttl=64 time=80.9 ms
64 bytes from 10.0.0.4: icmp_seq=22 ttl=64 time=81.3 ms
64 bytes from 10.0.0.4: icmp_seq=23 ttl=64 time=81.2 ms
64 bytes from 10.0.0.4: icmp_seq=24 ttl=64 time=80.5 ms
64 bytes from 10.0.0.4: icmp_seq=25 ttl=64 time=80.3 ms
64 bytes from 10.0.0.4: icmp_seq=26 ttl=64 time=80.4 ms
64 bytes from 10.0.0.4: icmp_seq=27 ttl=64 time=80.2 ms
64 bytes from 10.0.0.4: icmp_seq=28 ttl=64 time=80.6 ms
64 bytes from 10.0.0.4: icmp_seq=29 ttl=64 time=80.5 ms
64 bytes from 10.0.0.4: icmp_seq=30 ttl=64 time=80.5 ms
64 bytes from 10.0.0.4: icmp_seq=31 ttl=64 time=80.7 ms
64 bytes from 10.0.0.4: icmp_seq=32 ttl=64 time=80.4 ms
64 bytes from 10.0.0.4: icmp_seq=33 ttl=64 time=81.1 ms
64 bytes from 10.0.0.4: icmp_seq=34 ttl=64 time=81.1 ms
64 bytes from 10.0.0.4: icmp_seq=35 ttl=64 time=80.4 ms
64 bytes from 10.0.0.4: icmp_seq=36 ttl=64 time=80.6 ms
64 bytes from 10.0.0.4: icmp_seq=37 ttl=64 time=80.7 ms
64 bytes from 10.0.0.4: icmp_seq=38 ttl=64 time=80.5 ms
64 bytes from 10.0.0.4: icmp_seq=39 ttl=64 time=80.7 ms
64 bytes from 10.0.0.4: icmp_seq=40 ttl=64 time=80.5 ms

--- 10.0.0.4 ping statistics ---
40 packets transmitted, 38 received, 5% packet loss, time 39102ms
rtt min/avg/max/mdev = 80.240/112.045/1175.841/175.517 ms, pipe 2
```

Figura 11: 40 paquetes ping desde host 3 a host 4

Luego de haber enviado distintas cantidades de ping considerable entre el host 3 y el host 4, se puede apreciar que al enviar 40 ping, se logra obtener un error del 5 %.

2.2. Actividad 2

En esta actividad se continua con la topología creada en la actividad 1, en la cual se monta un servidor HTTP en el host 1 y se ejecuta una petición GET desde el host 2 hacia el servidor recientemente creado. También se capturan los paquetes del trafico de red, comprobando efectivamente el requerimiento GET con las IPs 10.0.0.1 y 10.0.0.2, las cuales son las direcciones del host 1 y host 2 respectivamente.

```

mininet> h1 python3 -m http.server 80 &
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
mininet> h2 wget -o - - h1
--2023-09-23 20:10:53-- http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 469 [text/html]
Saving to: 'index.html'

OK                               100% 51.7M=0s

2023-09-23 20:10:53 (51.7 MB/s) - 'index.html' saved [469/469]

```

Figura 12: Inicio servidor HTTP host 1 y Petición GET desde host 2 hacia el servidor.

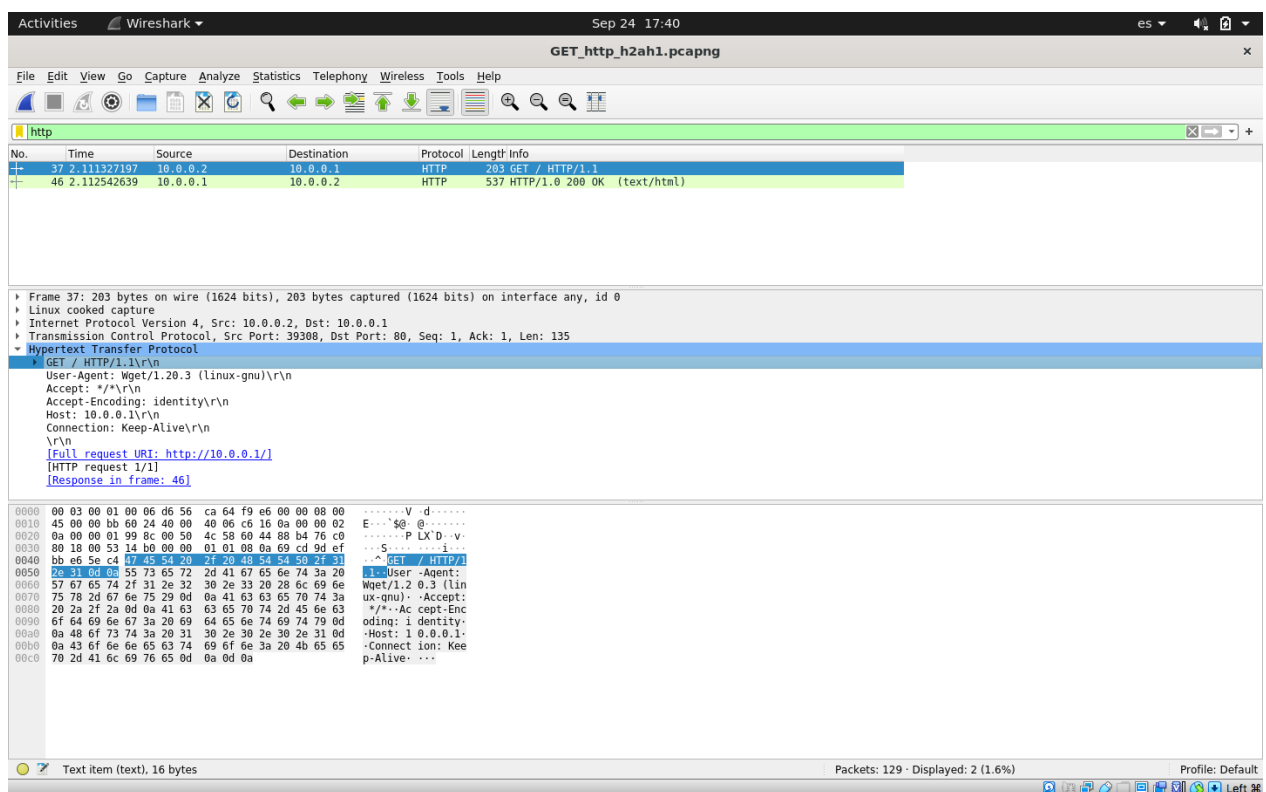


Figura 13: Captura Wireshark con filtro HTTP.

2.3. Actividad 3

2.3.1. Ejecución script nat.py y error nameserver

Para esta actividad se ocupa el script *nat.py* que crea una red con topología *tree* con conexión a internet gracias al uso del protocolo NAT. El script está en el directorio `/mininet/examples/`.


```

mininet@mininet-vm:~/mininet/examples$ 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> h1 ping -c 4 google.com
ping: google.com: Temporary failure in name resolution

```

Figura 14: Ejecución script nat.py y error de ping desde host 1 a www.google.com.

De la figura anterior se puede observar que existe un error de conexión al hacer ping desde algún host hacia una DNS. Esto es debido a que el servidor de nombres no esta funcionando correctamente.

2.3.2. Solución error nameserver

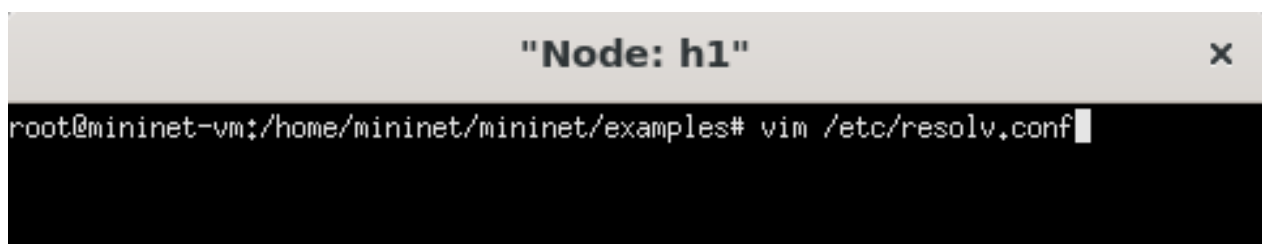
Una solución es cambiar el nameserver por defecto al DNS publico de google: 8.8.8.8. En este caso, se realiza a través de la terminal del host1 modificando el parámetro nameserver en el archivo resolv.conf ubicado en la ruta /etc/resolv.conf.

```

mininet> xterm h1
mininet>

```

Figura 15: Solución error nameserver.

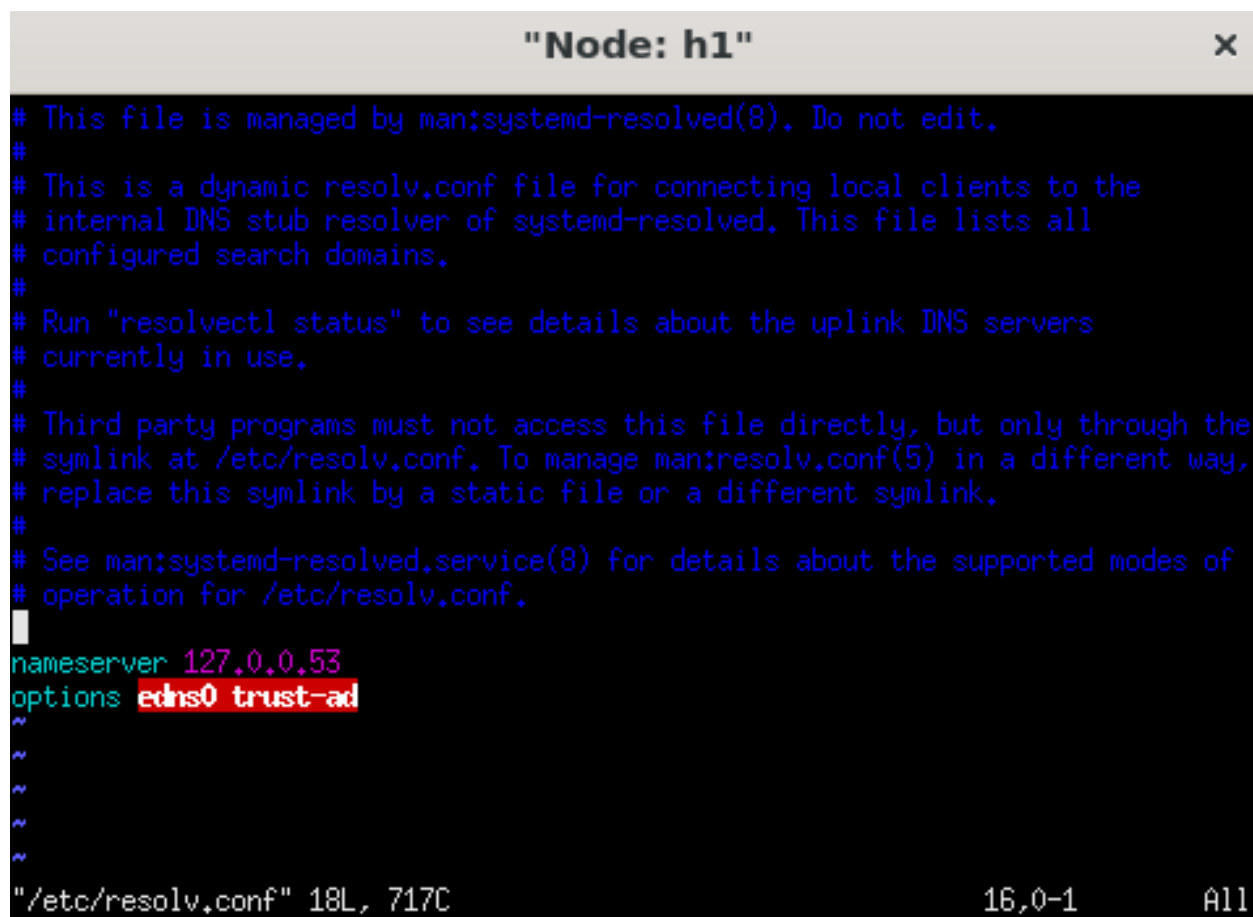


```

"Node: h1"
root@mininet-vm:/home/mininet/mininet/examples# vim /etc/resolv.conf

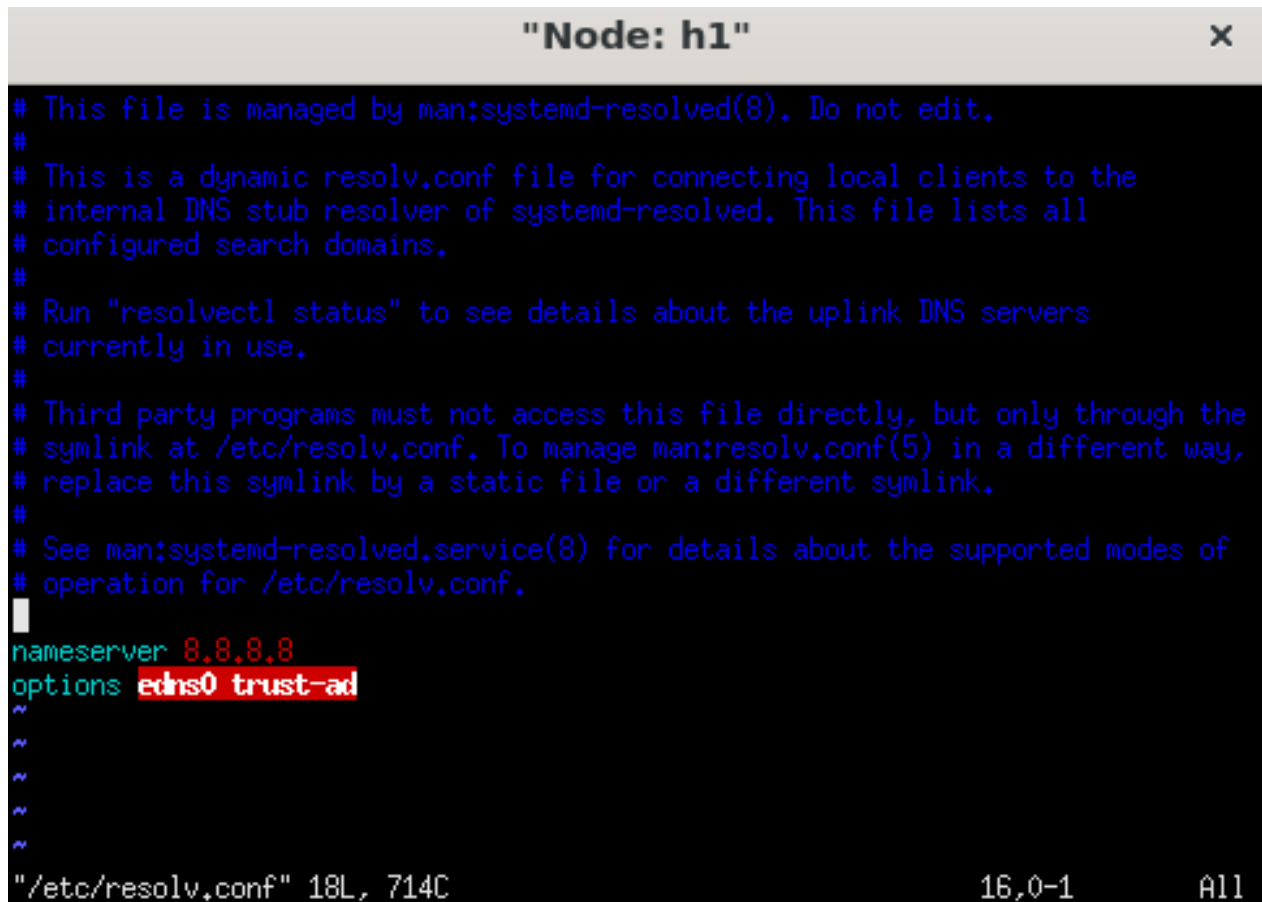
```

Figura 16: Solución error nameserver.



```
"Node: h1" x
# This file is managed by man:systemd-resolved(8). Do not edit.
#
# This is a dynamic resolv.conf file for connecting local clients to the
# internal DNS stub resolver of systemd-resolved. This file lists all
# configured search domains.
#
# Run "resolvectl status" to see details about the uplink DNS servers
# currently in use.
#
# Third party programs must not access this file directly, but only through the
# symlink at /etc/resolv.conf. To manage man:resolv.conf(5) in a different way,
# replace this symlink by a static file or a different symlink.
#
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.
nameserver 127.0.0.53
options edns0 trust-ad
~
~
~
~
~
"/etc/resolv.conf" 18L, 717C 16,0-1 All
```

Figura 17: Solución error nameserver.



```
"Node: h1"
# This file is managed by man:systemd-resolved(8). Do not edit.
#
# This is a dynamic resolv.conf file for connecting local clients to the
# internal DNS stub resolver of systemd-resolved. This file lists all
# configured search domains.
#
# Run "resolvectl status" to see details about the uplink DNS servers
# currently in use.
#
# Third party programs must not access this file directly, but only through the
# symlink at /etc/resolv.conf. To manage man:resolv.conf(5) in a different way,
# replace this symlink by a static file or a different symlink.
#
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.
nameserver 8.8.8.8
options edns0 trust-ad
~
~
~
~
~
"/etc/resolv.conf" 18L, 714C 16,0-1 All
```

Figura 18: Solución error nameserver.

2.3.3. Pruebas ping desde hosts hacia un DNS

Luego de haber configurado el nameserver, se prueba nuevamente haciendo ping desde cada uno de los host de la red hacia el DNS de amazon: `www.amazon.com`, y se verifican las IPs correspondiente través de la captura de tráfico de red hecha por Wireshark.

```

mininet> h1 ping -c 4 www.amazon.com
PING d3ag4hukkh62yn.cloudfront.net (13.227.207.168) 56(84) bytes of data.
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=1 ttl=6
1 time=9.28 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=2 ttl=6
1 time=7.35 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=3 ttl=6
1 time=8.51 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=4 ttl=6
1 time=8.21 ms

--- d3ag4hukkh62yn.cloudfront.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 7.345/8.335/9.283/0.693 ms
mininet>

```

Figura 19: Prueba ping desde host 1 a DNS de amazon.

The image shows a Wireshark packet capture window titled "pruebadnsamazon.pcapng". The filter bar shows "icmp or arp or dns && ip.addr==10.0.0.1". The packet list shows 31 packets. The packet details pane shows the selected packet (No. 31) as an ICMP Echo (ping) request from 10.0.0.1 to 8.8.8.8. The packet bytes pane shows the raw data of the packet.

No.	Time	Source	Destination	Protocol	Length	Info
24	8.043925487	10.0.0.1	8.8.8.8	DNS	87	Standard query 0x4c15 A www.amazon.com OPT
25	8.046882747	10.0.0.1	8.8.8.8	OpenFL	171	Type: OFPT_PACKET_IN
28	8.046993760	10.0.0.1	8.8.8.8	OpenFL	177	Type: OFPT_PACKET_OUT
30	8.048454342	10.0.0.1	8.8.8.8	DNS	87	Standard query 0x4c15 A www.amazon.com OPT
31	8.04857467	10.0.0.1	8.8.8.8	DNS	87	Standard query 0x4c15 A www.amazon.com OPT
34	8.220402908	8.8.8.8	10.0.0.1	DNS	216	Standard query response 0x4c15 A www.amazon.com CNAME tp.47cf-
35	8.220408026	8.8.8.8	10.0.0.1	DNS	216	Standard query response 0x4c15 A www.amazon.com CNAME tp.47cf-
36	8.220975463	8.8.8.8	10.0.0.1	OpenFL	300	Type: OFPT_PACKET_IN
39	8.221304261	8.8.8.8	10.0.0.1	OpenFL	306	Type: OFPT_PACKET_OUT
41	8.221516006	8.8.8.8	10.0.0.1	DNS	216	Standard query response 0x4c15 A www.amazon.com CNAME tp.47cf-
42	8.222012009	10.0.0.1	23.222.253.234	ICMP	100	Echo (ping) request id=0x342b, seq=1/256, ttl=64 (no respons...
43	8.235401754	10.0.0.1	23.222.253.234	OpenFL	184	Type: OFPT_PACKET_IN
46	8.235629607	10.0.0.1	23.222.253.234	OpenFL	190	Type: OFPT_PACKET_OUT
48	8.235836433	10.0.0.1	23.222.253.234	ICMP	100	Echo (ping) request id=0x342b, seq=1/256, ttl=64 (no respons...
49	8.235840370	10.0.0.1	23.222.253.234	ICMP	100	Echo (ping) request id=0x342b, seq=1/256, ttl=64 (reply in 5...
52	8.242201676	23.222.253.234	10.0.0.1	ICMP	100	Echo (ping) reply id=0x342b, seq=1/256, ttl=61 (request in...
53	8.242208072	23.222.253.234	10.0.0.1	ICMP	100	Echo (ping) reply id=0x342b, seq=1/256, ttl=61
54	8.246877454	23.222.253.234	10.0.0.1	OpenFL	184	Type: OFPT_PACKET_IN
56	8.246998935	23.222.253.234	10.0.0.1	OpenFL	190	Type: OFPT_PACKET_OUT
58	8.249530589	23.222.253.234	10.0.0.1	ICMP	100	Echo (ping) reply id=0x342b, seq=1/256, ttl=61
59	8.249781933	10.0.0.1	8.8.8.8	DNS	100	Standard query 0x7848 PTR 234.253.222.23.in-addr.arpa OPT
60	8.255213938	10.0.0.1	8.8.8.8	OpenFL	184	Type: OFPT_PACKET_IN
62	8.255293016	10.0.0.1	8.8.8.8	OpenFL	190	Type: OFPT_PACKET_OUT
64	8.259355730	10.0.0.1	8.8.8.8	DNS	100	Standard query 0x7848 PTR 234.253.222.23.in-addr.arpa OPT
65	8.259358883	10.0.0.1	8.8.8.8	DNS	100	Standard query 0x7848 PTR 234.253.222.23.in-addr.arpa OPT
68	8.266252788	8.8.8.8	10.0.0.1	DNS	166	Standard query response 0x7848 PTR 234.253.222.23.in-addr.arpa...
69	8.266258587	8.8.8.8	10.0.0.1	DNS	166	Standard query response 0x7848 PTR 234.253.222.23.in-addr.arpa...
70	8.266718506	8.8.8.8	10.0.0.1	OpenFL	250	Type: OFPT_PACKET_IN
72	8.266854023	8.8.8.8	10.0.0.1	OpenFL	256	Type: OFPT_PACKET_OUT
74	8.266996359	8.8.8.8	10.0.0.1	DNS	166	Standard query response 0x7848 PTR 234.253.222.23.in-addr.arpa...
75	9.223295491	10.0.0.1	23.222.253.234	ICMP	100	Echo (ping) request id=0x342b, seq=2/512, ttl=64 (no respons...
76	9.223542955	10.0.0.1	23.222.253.234	ICMP	100	Echo (ping) request id=0x342b, seq=2/512, ttl=64 (no respons...
77	9.223545964	10.0.0.1	23.222.253.234	ICMP	100	Echo (ping) request id=0x342b, seq=2/512, ttl=64 (reply in 8...
80	9.231488575	23.222.253.234	10.0.0.1	ICMP	100	Echo (ping) reply id=0x342b, seq=2/512, ttl=61 (request in...
81	9.231492619	23.222.253.234	10.0.0.1	ICMP	100	Echo (ping) reply id=0x342b, seq=2/512, ttl=61

Figura 20: Captura Wireshark con filtros icmp, arp, DNS y dirección IP de host 1.

```

mininet> h2 ping -c 4 www.amazon.com
PING d3ag4hukkh62yn.cloudfront.net (13.227.207.168) 56(84) bytes of data.
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=1 ttl=6
1 time=10.5 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=2 ttl=6
1 time=7.32 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=3 ttl=6
1 time=8.61 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=4 ttl=6
1 time=8.44 ms

--- d3ag4hukkh62yn.cloudfront.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 7.320/8.704/10.450/1.122 ms
mininet>

```

Figura 21: Prueba ping desde host 2 a DNS de amazon.

No.	Time	Source	Destination	Protocol	Length	Info
143	23.654417397	10.0.0.2	8.8.8.8	DNS	87	Standard query 0x7c1b A www.amazon.com OPT
144	23.654952289	10.0.0.2	8.8.8.8	OpenFL	171	Type: OFPT_PACKET_IN
147	23.655015664	10.0.0.2	8.8.8.8	OpenFL	177	Type: OFPT_PACKET_OUT
149	23.659191695	10.0.0.2	8.8.8.8	DNS	87	Standard query 0x7c1b A www.amazon.com OPT
150	23.659198593	10.0.0.2	8.8.8.8	DNS	87	Standard query 0x7c1b A www.amazon.com OPT
153	23.771691172	8.8.8.8	10.0.0.2	DNS	182	Standard query response 0x7c1b A www.amazon.com CNAME tp.47cf_
154	23.771696241	8.8.8.8	10.0.0.2	DNS	182	Standard query response 0x7c1b A www.amazon.com CNAME tp.47cf_
155	23.772208822	8.8.8.8	10.0.0.2	OpenFL	266	Type: OFPT_PACKET_IN
158	23.772470630	8.8.8.8	10.0.0.2	OpenFL	272	Type: OFPT_PACKET_OUT
160	23.772652113	8.8.8.8	10.0.0.2	DNS	182	Standard query response 0x7c1b A www.amazon.com CNAME tp.47cf_
161	23.773205207	10.0.0.2	3.162.214.136	ICMP	100	Echo (ping) request id=0x3431, seq=1/256, ttl=64 (no respons...
162	23.774041410	10.0.0.2	3.162.214.136	OpenFL	184	Type: OFPT_PACKET_IN
165	23.774145561	10.0.0.2	3.162.214.136	OpenFL	190	Type: OFPT_PACKET_OUT
167	23.776093223	10.0.0.2	3.162.214.136	ICMP	100	Echo (ping) request id=0x3431, seq=1/256, ttl=64 (no respons...
168	23.776099181	10.0.0.2	3.162.214.136	ICMP	100	Echo (ping) request id=0x3431, seq=1/256, ttl=64 (reply in l...
171	23.782483248	3.162.214.136	10.0.0.2	ICMP	100	Echo (ping) reply id=0x3431, seq=1/256, ttl=61 (request in...
172	23.782486325	3.162.214.136	10.0.0.2	ICMP	100	Echo (ping) reply id=0x3431, seq=1/256, ttl=61
173	23.782749387	3.162.214.136	10.0.0.2	OpenFL	184	Type: OFPT_PACKET_IN
175	23.782850736	3.162.214.136	10.0.0.2	OpenFL	190	Type: OFPT_PACKET_OUT
177	23.782975181	3.162.214.136	10.0.0.2	ICMP	100	Echo (ping) reply id=0x3431, seq=1/256, ttl=61
178	23.783201650	10.0.0.2	8.8.8.8	DNS	99	Standard query 0x6dbd PTR 136.214.162.3.in-addr.arpa OPT
179	23.783501430	10.0.0.2	8.8.8.8	OpenFL	183	Type: OFPT_PACKET_IN
181	23.783556978	10.0.0.2	8.8.8.8	OpenFL	189	Type: OFPT_PACKET_OUT
183	23.783789520	10.0.0.2	8.8.8.8	DNS	99	Standard query 0x6dbd PTR 136.214.162.3.in-addr.arpa OPT
184	23.783835471	10.0.0.2	8.8.8.8	DNS	99	Standard query 0x6dbd PTR 136.214.162.3.in-addr.arpa OPT
187	23.790256392	8.8.8.8	10.0.0.2	DNS	156	Standard query response 0x6dbd PTR 136.214.162.3.in-addr.arpa...
188	23.790259992	8.8.8.8	10.0.0.2	DNS	156	Standard query response 0x6dbd PTR 136.214.162.3.in-addr.arpa...
189	23.791135372	8.8.8.8	10.0.0.2	OpenFL	240	Type: OFPT_PACKET_IN
191	23.791306506	8.8.8.8	10.0.0.2	OpenFL	246	Type: OFPT_PACKET_OUT
193	23.791713559	8.8.8.8	10.0.0.2	DNS	156	Standard query response 0x6dbd PTR 136.214.162.3.in-addr.arpa...
194	24.775552920	10.0.0.2	3.162.214.136	ICMP	100	Echo (ping) request id=0x3431, seq=2/512, ttl=64 (no respons...
195	24.775908925	10.0.0.2	3.162.214.136	ICMP	100	Echo (ping) request id=0x3431, seq=2/512, ttl=64 (no respons...
196	24.775913066	10.0.0.2	3.162.214.136	ICMP	100	Echo (ping) request id=0x3431, seq=2/512, ttl=64 (reply in l...
199	24.785266908	3.162.214.136	10.0.0.2	ICMP	100	Echo (ping) reply id=0x3431, seq=2/512, ttl=61 (request in...
200	24.785270750	3.162.214.136	10.0.0.2	ICMP	100	Echo (ping) reply id=0x3431, seq=2/512, ttl=61

Frame 143: 87 bytes on wire (696 bits), 87 bytes captured (696 bits) on interface any, id 0

```

0000 00 03 00 01 00 06 fe 5e 28 b9 8e 78 00 a8 08 00 .....^(..x....
0010 45 00 00 47 54 8b 40 00 40 11 cc 09 0a 00 00 02 E..GT.@.@.....
0020 08 08 08 08 96 4a 00 35 00 33 1a 56 7c 1b 01 20 .....J-5-3V|...
0030 00 01 00 00 00 00 00 01 03 77 77 77 06 61 6d 61 .....wwwama

```

Figura 22: Captura Wireshark con filtros icmp, arp, DNS y dirección IP de host 2.

```

mininet> h3 ping -c 4 www.amazon.com
PING e15316.dsca.akamaiedge.net (23.222.253.234) 56(84) bytes of data.
64 bytes from a23-222-253-234.deploy.static.akamaitechnologies.com (23.222.253.234): icmp_seq=1 ttl=61 time=13.6 ms
64 bytes from a23-222-253-234.deploy.static.akamaitechnologies.com (23.222.253.234): icmp_seq=2 ttl=61 time=7.21 ms
64 bytes from a23-222-253-234.deploy.static.akamaitechnologies.com (23.222.253.234): icmp_seq=3 ttl=61 time=7.51 ms
64 bytes from a23-222-253-234.deploy.static.akamaitechnologies.com (23.222.253.234): icmp_seq=4 ttl=61 time=6.35 ms

--- e15316.dsca.akamaiedge.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 6.346/8.663/13.588/2.874 ms
mininet>

```

Figura 23: Prueba ping desde host 3 a DNS de amazon.

No.	Time	Source	Destination	Protocol	Length	Info
254	33.98389888	10.0.0.3	8.8.8.8	DNS	87	Standard query 0xb1ec A www.amazon.com OPT
255	33.988714514	10.0.0.3	8.8.8.8	OpenFL	171	Type: OFPT_PACKET_IN
258	33.988778331	10.0.0.3	8.8.8.8	OpenFL	177	Type: OFPT_PACKET_OUT
260	33.989162747	10.0.0.3	8.8.8.8	DNS	87	Standard query 0xb1ec A www.amazon.com OPT
261	33.989164525	10.0.0.3	8.8.8.8	DNS	87	Standard query 0xb1ec A www.amazon.com OPT
264	34.103899722	8.8.8.8	10.0.0.3	DNS	182	Standard query response 0xb1ec A www.amazon.com CNAME tp.47cf-
265	34.103817915	8.8.8.8	10.0.0.3	DNS	182	Standard query response 0xb1ec A www.amazon.com CNAME tp.47cf-
266	34.104382360	8.8.8.8	10.0.0.3	OpenFL	266	Type: OFPT_PACKET_IN
269	34.104690553	8.8.8.8	10.0.0.3	OpenFL	272	Type: OFPT_PACKET_OUT
271	34.104973271	8.8.8.8	10.0.0.3	DNS	182	Standard query response 0xb1ec A www.amazon.com CNAME tp.47cf-
272	34.105426412	10.0.0.3	3.162.214.136	ICMP	100	Echo (ping) request id=0x3436, seq=1/256, ttl=64 (no respons...
273	34.106792219	10.0.0.3	3.162.214.136	OpenFL	184	Type: OFPT_PACKET_IN
276	34.106891027	10.0.0.3	3.162.214.136	OpenFL	190	Type: OFPT_PACKET_OUT
278	34.108878430	10.0.0.3	3.162.214.136	ICMP	100	Echo (ping) request id=0x3436, seq=1/256, ttl=64 (no respons...
279	34.108881964	10.0.0.3	3.162.214.136	ICMP	100	Echo (ping) request id=0x3436, seq=1/256, ttl=64 (reply in 2...
282	34.115680478	3.162.214.136	10.0.0.3	ICMP	100	Echo (ping) reply id=0x3436, seq=1/256, ttl=61 (request in...
283	34.115684390	3.162.214.136	10.0.0.3	ICMP	100	Echo (ping) reply id=0x3436, seq=1/256, ttl=61
284	34.116280297	3.162.214.136	10.0.0.3	OpenFL	184	Type: OFPT_PACKET_IN
286	34.116411949	3.162.214.136	10.0.0.3	OpenFL	190	Type: OFPT_PACKET_OUT
288	34.116567733	3.162.214.136	10.0.0.3	ICMP	100	Echo (ping) reply id=0x3436, seq=1/256, ttl=61
289	34.116837990	10.0.0.3	8.8.8.8	DNS	99	Standard query 0x73e8 PTR 136.214.162.3.in-addr.arpa OPT
290	34.117180114	10.0.0.3	8.8.8.8	OpenFL	183	Type: OFPT_PACKET_IN
292	34.117262360	10.0.0.3	8.8.8.8	OpenFL	189	Type: OFPT_PACKET_OUT
294	34.122951371	10.0.0.3	8.8.8.8	DNS	99	Standard query 0x73e8 PTR 136.214.162.3.in-addr.arpa OPT
295	34.122954731	10.0.0.3	8.8.8.8	DNS	156	Standard query response 0x73e8 PTR 136.214.162.3.in-addr.arpa...
298	34.130371809	8.8.8.8	10.0.0.3	DNS	156	Standard query response 0x73e8 PTR 136.214.162.3.in-addr.arpa...
299	34.130375327	8.8.8.8	10.0.0.3	DNS	156	Standard query response 0x73e8 PTR 136.214.162.3.in-addr.arpa...
300	34.130659803	8.8.8.8	10.0.0.3	OpenFL	240	Type: OFPT_PACKET_IN
302	34.130844591	8.8.8.8	10.0.0.3	OpenFL	246	Type: OFPT_PACKET_OUT
304	34.130997500	8.8.8.8	10.0.0.3	DNS	156	Standard query response 0x73e8 PTR 136.214.162.3.in-addr.arpa...
305	35.106544749	10.0.0.3	3.162.214.136	ICMP	100	Echo (ping) request id=0x3436, seq=2/512, ttl=64 (no respons...
306	35.106905322	10.0.0.3	3.162.214.136	ICMP	100	Echo (ping) request id=0x3436, seq=2/512, ttl=64 (no respons...
307	35.106909308	10.0.0.3	3.162.214.136	ICMP	100	Echo (ping) request id=0x3436, seq=2/512, ttl=64 (reply in 3...
310	35.115040746	3.162.214.136	10.0.0.3	ICMP	100	Echo (ping) reply id=0x3436, seq=2/512, ttl=61 (request in...
311	35.115043813	3.162.214.136	10.0.0.3	ICMP	100	Echo (ping) reply id=0x3436, seq=2/512, ttl=61

Figura 24: Captura Wireshark con filtros icmp, arp, DNS y dirección IP de host 3.

```

mininet> h4 ping -c 4 www.amazon.com
PING d3ag4hukkh62yn.cloudfront.net (13.227.207.168) 56(84) bytes of data.
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=1 ttl=6
1 time=11.7 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=2 ttl=6
1 time=9.23 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=3 ttl=6
1 time=8.01 ms
64 bytes from server-13-227-207-168.scl50.r.cloudfront.net (13.227.207.168): icmp_seq=4 ttl=6
1 time=8.48 ms

--- d3ag4hukkh62yn.cloudfront.net ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 8.006/9.364/11.744/1.440 ms
mininet>

```

Figura 25: Prueba ping desde host 4 a DNS de amazon.

No.	Time	Source	Destination	Protocol	Length	Info
365	41.116759930	10.0.0.4	8.8.8.8	DNS	87	Standard query 0xd22e A www.amazon.com OPT
366	41.117057734	10.0.0.4	8.8.8.8	OpenFL	171	Type: OFPT_PACKET_IN
369	41.117116828	10.0.0.4	8.8.8.8	OpenFL	177	Type: OFPT_PACKET_OUT
371	41.121482026	10.0.0.4	8.8.8.8	DNS	87	Standard query 0xd22e A www.amazon.com OPT
372	41.121487589	10.0.0.4	8.8.8.8	DNS	87	Standard query 0xd22e A www.amazon.com OPT
375	41.368871560	8.8.8.8	10.0.0.4	DNS	187	Standard query response 0xd22e A www.amazon.com CNAME tp.47cf_
376	41.368877871	8.8.8.8	10.0.0.4	DNS	187	Standard query response 0xd22e A www.amazon.com CNAME tp.47cf_
377	41.369497283	8.8.8.8	10.0.0.4	OpenFL	271	Type: OFPT_PACKET_IN
380	41.369839440	8.8.8.8	10.0.0.4	OpenFL	277	Type: OFPT_PACKET_OUT
382	41.370858063	8.8.8.8	10.0.0.4	DNS	187	Standard query response 0xd22e A www.amazon.com CNAME tp.47cf_
383	41.370543578	10.0.0.4	162.219.225.118	ICMP	100	Echo (ping) request id=0x3439, seq=1/256, ttl=64 (no respons...
384	41.371806084	10.0.0.4	162.219.225.118	OpenFL	184	Type: OFPT_PACKET_IN
387	41.371904423	10.0.0.4	162.219.225.118	OpenFL	190	Type: OFPT_PACKET_OUT
389	41.373155120	10.0.0.4	162.219.225.118	ICMP	100	Echo (ping) request id=0x3439, seq=1/256, ttl=64 (no respons...
390	41.373160261	10.0.0.4	162.219.225.118	ICMP	100	Echo (ping) request id=0x3439, seq=1/256, ttl=61 (request in...
393	41.381305563	162.219.225.118	10.0.0.4	ICMP	100	Echo (ping) reply id=0x3439, seq=1/256, ttl=61
394	41.381309595	162.219.225.118	10.0.0.4	ICMP	100	Echo (ping) reply id=0x3439, seq=1/256, ttl=61
395	41.381788825	162.219.225.118	10.0.0.4	OpenFL	184	Type: OFPT_PACKET_IN
398	41.381952345	162.219.225.118	10.0.0.4	OpenFL	190	Type: OFPT_PACKET_OUT
400	41.382120180	162.219.225.118	10.0.0.4	ICMP	100	Echo (ping) reply id=0x3439, seq=1/256, ttl=61
401	41.382420500	10.0.0.4	8.8.8.8	DNS	101	Standard query 0x147c PTR 118.225.219.162.in-addr.arpa OPT
402	41.382980944	10.0.0.4	8.8.8.8	OpenFL	185	Type: OFPT_PACKET_IN
405	41.383179226	10.0.0.4	8.8.8.8	OpenFL	191	Type: OFPT_PACKET_OUT
407	41.389053548	10.0.0.4	8.8.8.8	DNS	101	Standard query 0x147c PTR 118.225.219.162.in-addr.arpa OPT
408	41.389061649	10.0.0.4	8.8.8.8	DNS	101	Standard query 0x147c PTR 118.225.219.162.in-addr.arpa OPT
411	41.394345271	8.8.8.8	10.0.0.4	DNS	155	Standard query response 0x147c No such name PTR 118.225.219.1...
412	41.394347997	8.8.8.8	10.0.0.4	DNS	155	Standard query response 0x147c No such name PTR 118.225.219.1...
413	41.394621534	8.8.8.8	10.0.0.4	OpenFL	239	Type: OFPT_PACKET_IN
416	41.395049746	8.8.8.8	10.0.0.4	OpenFL	245	Type: OFPT_PACKET_OUT
418	41.395270985	8.8.8.8	10.0.0.4	DNS	155	Standard query response 0x147c No such name PTR 118.225.219.1...
419	42.373595709	10.0.0.4	162.219.225.118	ICMP	100	Echo (ping) request id=0x3439, seq=2/512, ttl=64 (no respons...
420	42.373885190	10.0.0.4	162.219.225.118	ICMP	100	Echo (ping) request id=0x3439, seq=2/512, ttl=64 (no respons...
421	42.373888614	10.0.0.4	162.219.225.118	ICMP	100	Echo (ping) request id=0x3439, seq=2/512, ttl=64 (reply in 4...
424	42.381304897	162.219.225.118	10.0.0.4	ICMP	100	Echo (ping) reply id=0x3439, seq=2/512, ttl=61 (request in...
425	42.381307674	162.219.225.118	10.0.0.4	ICMP	100	Echo (ping) reply id=0x3439, seq=2/512, ttl=61

Figura 26: Captura Wireshark con filtros icmp, arp, DNS y dirección IP de host 4.

2.3.4. Ejecución de servidor HTTP en host 2 e ingreso pagina web desde host 3

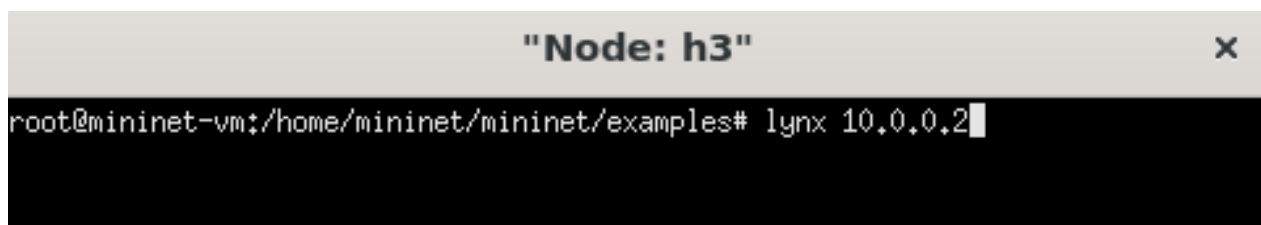
Después de haber verificado las respuesta a ping, se captura el tráfico de red con Wireshark y se inicia un servidor HTTP en el host 2. Luego se utiliza el browser de modo texto: Lynx, desde la terminal del host 3, esto con el objetivo de ingresar a la pagina web del servidor HTTP con la dirección 10.0.0.2.

```
mininet> h2 python3 -m http.server 80 &
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Figura 27: Inicio servidor HTTP en host 2.

```
mininet> xterm h3
mininet> █
```

Figura 28: Comando para abrir terminal host 3.



```
"Node: h3"
root@mininet-vm:/home/mininet/mininet/examples# lynx 10.0.0.2
```

Figura 29: Ingreso desde host 3 con lynx con dirección IP host 2.



```
"Node: h3"
Directory listing for /:
* _init_.py
* _pycache_/
* baresshd.py
* bind.py
* cluster.py
* clustercli.py
* clusterdemo.py
* clusterperf.py
* clusterstats.py
* consoles.py
* controllers.py
* controllers2.py
* controlnet.py
* cpu.py
* emptynet.py
* hwintf.py
* intfoptions.py
* limit.py
* linearbandwidth.py
* linuxrouter.py
* minidit.py
* mobility.py
* multilink.py
* multiping.py
* multipoll.py
* multitest.py
* nat.py
* natnet.py
* numberedports.py
* popen.py
* popenpoll.py
* RRDtool
* scratchnet.py
* scratchnetuser.py
* simpleperf.py
* sshd.py
* test/
* tree1024.py
* treeping64.py
* vlanhost.py

Legend: Use arrow keys to move. '?' for help. 'q' to quit. '<' to go back.
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H?help O?ptions P?rint Q?uit /?search [delete]=history list
```

Figura 30: Pagina web del servidor con lynx desde el host 3.

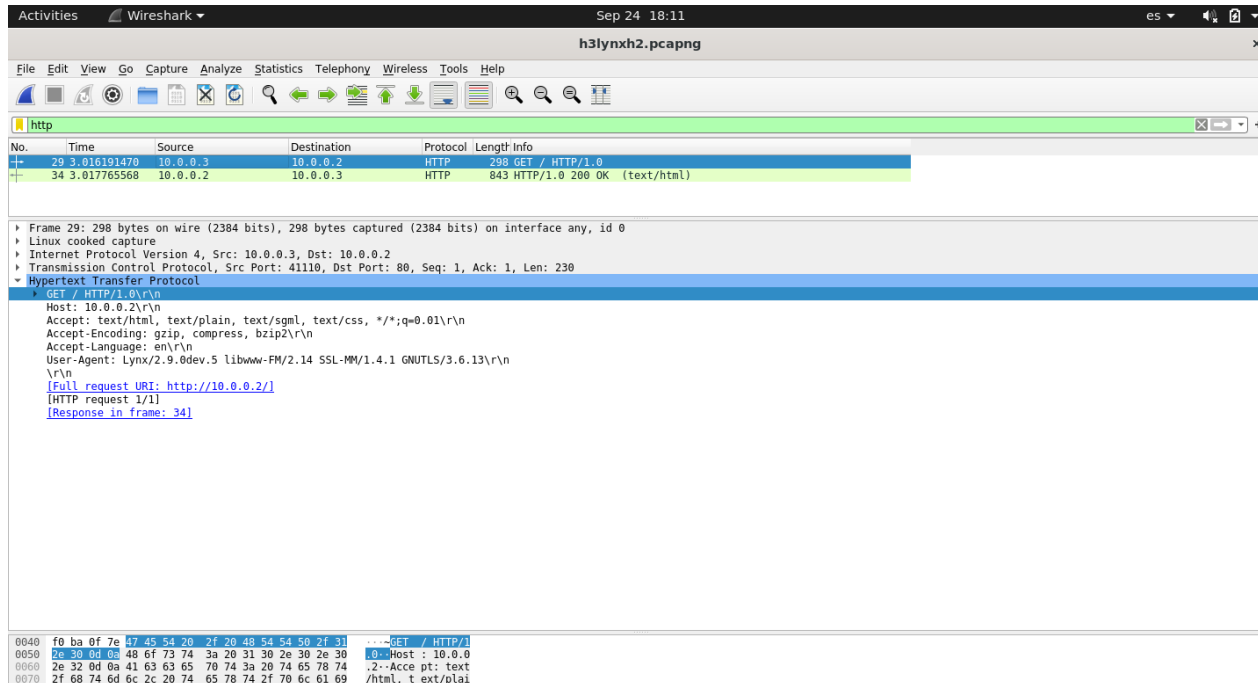


Figura 31: Captura Wireshark con filtros HTTP.

2.4. Actividad 4

En esta actividad se emula un enlace entre Chile y Australia, a través del siguiente script hecho en Python:

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def build( self ):
        "Create custom topo."

        # Add hosts and switches
        host_Chile = self.addHost( 'h1' )
        host_Australia = self.addHost( 'h2' )

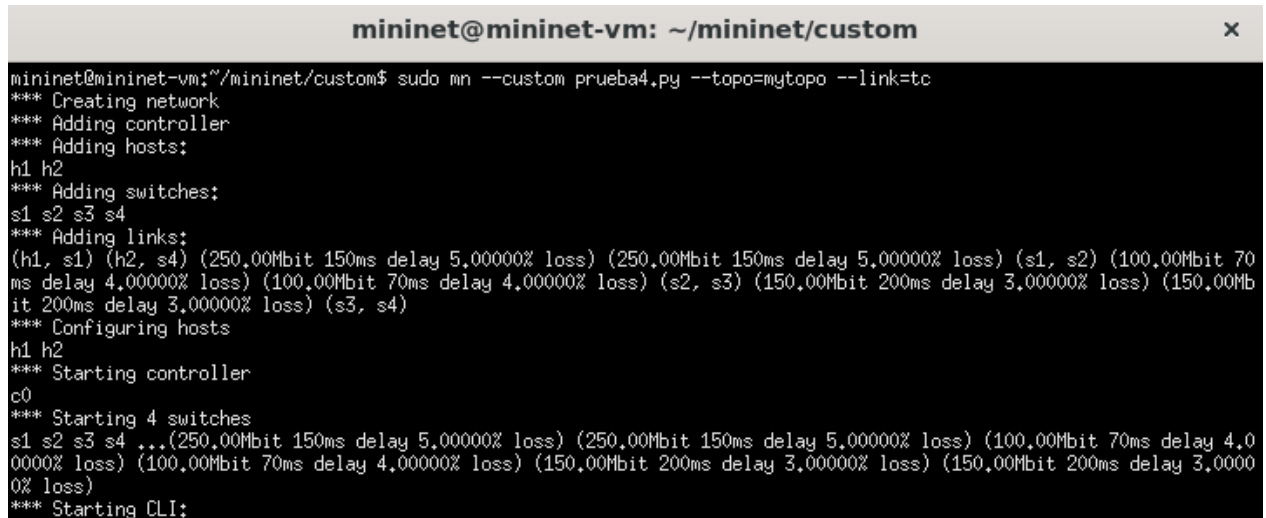
        Switch1 = self.addSwitch( 's1' )
        Switch2 = self.addSwitch( 's2' )
        Switch3 = self.addSwitch( 's3' )
        Switch4 = self.addSwitch( 's4' )

        # Add links
        self.addLink(host_Chile, Switch1)
        self.addLink(host_Australia, Switch4)

        self.addLink( Switch1, Switch2, bw=250, delay='150ms', loss=6)# especial
        self.addLink( Switch2, Switch3, bw=100, delay='70ms', loss=1)
        self.addLink( Switch3, Switch4, bw=150, delay='200ms', loss=3)# Especial

topos = { 'mytopo': ( lambda: MyTopo() ) }
```

Figura 32: Código Python actividad 4.



```
mininet@mininet-vm: ~/mininet/custom
mininet@mininet-vm:~/mininet/custom$ sudo mn --custom prueba4.py --topo=mytopo --link=tc
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s4) (250.00Mbit 150ms delay 5.00000% loss) (250.00Mbit 150ms delay 5.00000% loss) (s1, s2) (100.00Mbit 70
ms delay 4.00000% loss) (100.00Mbit 70ms delay 4.00000% loss) (s2, s3) (150.00Mbit 200ms delay 3.00000% loss) (150.00Mb
it 200ms delay 3.00000% loss) (s3, s4)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4 ... (250.00Mbit 150ms delay 5.00000% loss) (250.00Mbit 150ms delay 5.00000% loss) (100.00Mbit 70ms delay 4.0
0000% loss) (100.00Mbit 70ms delay 4.00000% loss) (150.00Mbit 200ms delay 3.00000% loss) (150.00Mbit 200ms delay 3.0000
0% loss)
*** Starting CLI:
```

Figura 33: Ejecución código Python actividad 4.

2.4.1. Iperf entre host Chile y host australia

En este parte se ejecuta el comando iperf en mininet entre los host Chile y Host Australia:



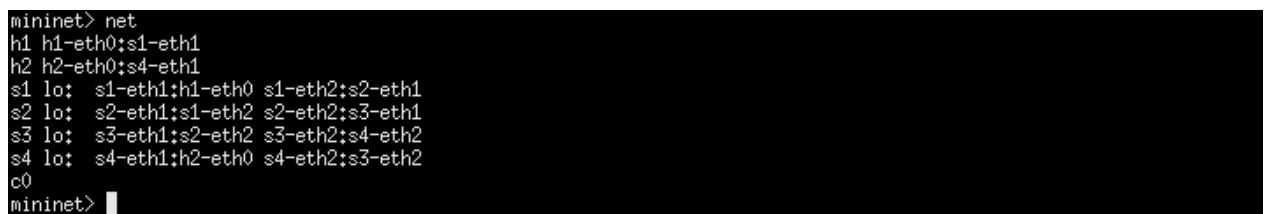
```
mininet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['50.8 Kbits/sec', '280 Kbits/sec']
mininet> net
```

Figura 34: Iperf entre host Chile y host Australia.

El resultado que se obtiene esta indicando que las tasas de transferencia son de "50,8 Kbits/sec" y "280 Kbits/sec", esto sugiere que la tasa de transferencia entre los hosts h1 y h2 es bastante baja. Esto puede ser principalmente debido a la configuración de la topología hecha, problemas de red o congestión, etc.

2.4.2. Transferencia de archivos usando FTP

Primeramente, se procede a capturar en Wireshark el trafico generado en la interfaz h1 y s1. Esta interfaz se verifica a través del comando net en mininet:



```
mininet> net
h1 h1-eth0:s1-eth1
h2 h2-eth0:s4-eth1
s1 lo: s1-eth1:h1-eth0 s1-eth2:s2-eth1
s2 lo: s2-eth1:s1-eth2 s2-eth2:s3-eth1
s3 lo: s3-eth1:s2-eth2 s3-eth2:s4-eth2
s4 lo: s4-eth1:h2-eth0 s4-eth2:s3-eth2
c0
mininet>
```

Figura 35: Comando net en Mininet.

Luego se ejecutan los siguientes pasos:

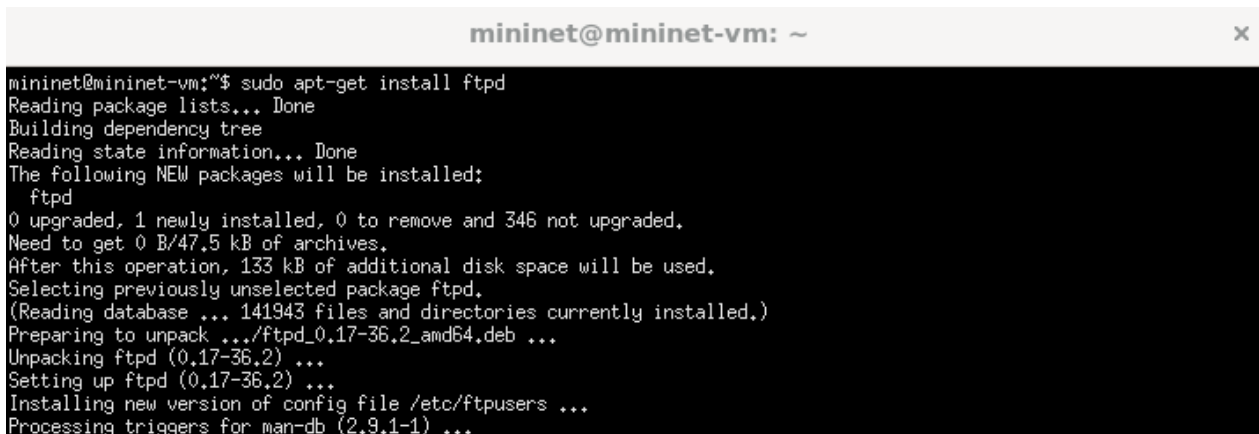
1. Creación archivo: En esta parte se crea el archivo goku.png de 20Mb a través del comando fallocate .



```
mininet@mininet-vm: ~
mininet@mininet-vm:~$ fallocate -l 20MiB goku.png
mininet@mininet-vm:~$ ls
Desktop  Downloads  mininet  oflops  openflow  pox  Templates
Documents goku.png  Music    oftest  Pictures  Public Videos
mininet@mininet-vm:~$
```

Figura 36: Creación archivo goku.png.

2. Instalación servidor FTP: En esta parte se procede a instalar ftpd.



```
mininet@mininet-vm: ~
mininet@mininet-vm:~$ sudo apt-get install ftpd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  ftpd
0 upgraded, 1 newly installed, 0 to remove and 346 not upgraded.
Need to get 0 B/47.5 kB of archives.
After this operation, 133 kB of additional disk space will be used.
Selecting previously unselected package ftpd.
(Reading database ... 141943 files and directories currently installed.)
Preparing to unpack .../ftpd_0.17-36.2_amd64.deb ...
Unpacking ftpd (0.17-36.2) ...
Setting up ftpd (0.17-36.2) ...
Installing new version of config file /etc/ftpusers ...
Processing triggers for man-db (2.9.1-1) ...
```

Figura 37: Instalación servidor FTP.

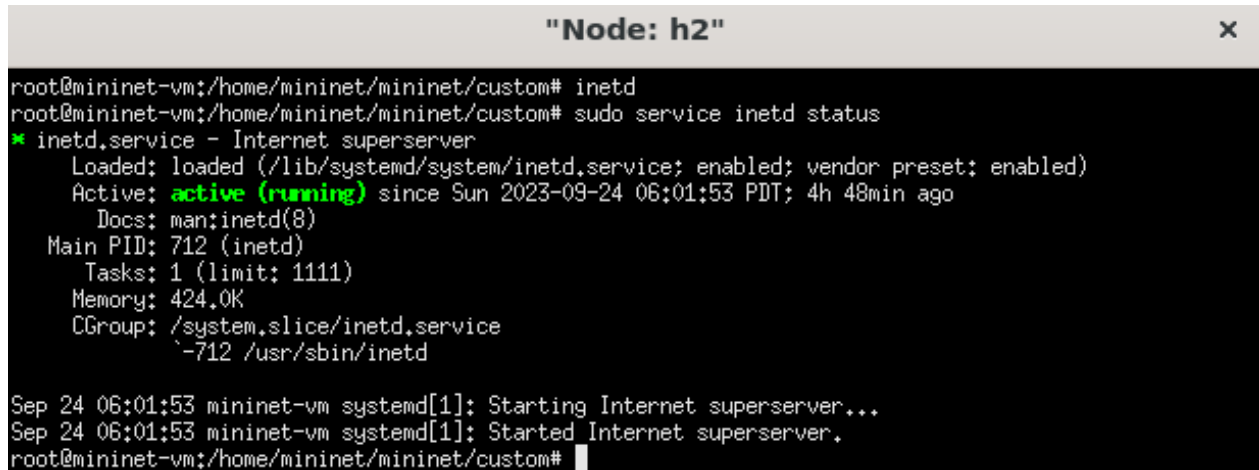
3. Xterm h1 y h2: Se inician simultáneamente las terminales xterm en los host de Chile y Australia.



```
mininet> xterm h1 h2
mininet>
```

Figura 38: Comando para abrir xterm en h1 y h2.

4. Inicio y estado servidor FTP h2: Se inicia un servidor FTP en el host de Australia usando inetd y se revisa el estado actual del servidor.



```

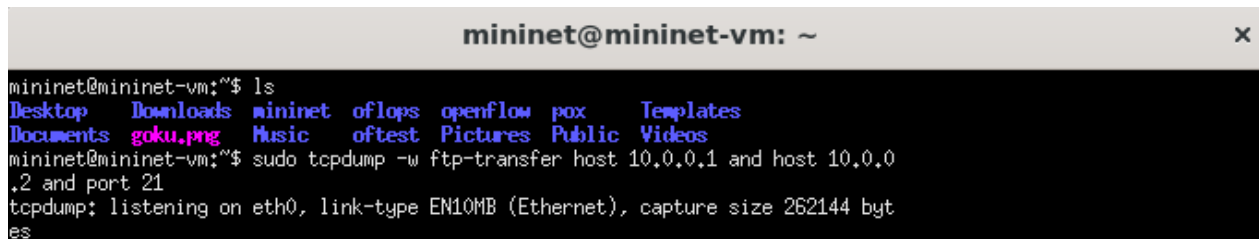
root@mininet-vm:/home/mininet/mininet/custom# inetd
root@mininet-vm:/home/mininet/mininet/custom# sudo service inetd status
* inetd.service - Internet superserver
   Loaded: loaded (/lib/systemd/system/inetd.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2023-09-24 06:01:53 PDT; 4h 48min ago
     Docs: man:inetd(8)
    Main PID: 712 (inetd)
      Tasks: 1 (limit: 1111)
     Memory: 424.0K
    CGroup: /system.slice/inetd.service
            └─712 /usr/sbin/inetd

Sep 24 06:01:53 mininet-vm systemd[1]: Starting Internet superserver...
Sep 24 06:01:53 mininet-vm systemd[1]: Started Internet superserver.
root@mininet-vm:/home/mininet/mininet/custom#

```

Figura 39: Inicio y estado del servidor FTP del host de Australia.

5. Captura TCPDUMP: Se captura el trafico de red de paquetes entre el host de Chile(h1) y el host de Australia(h2), en el puerto 21.



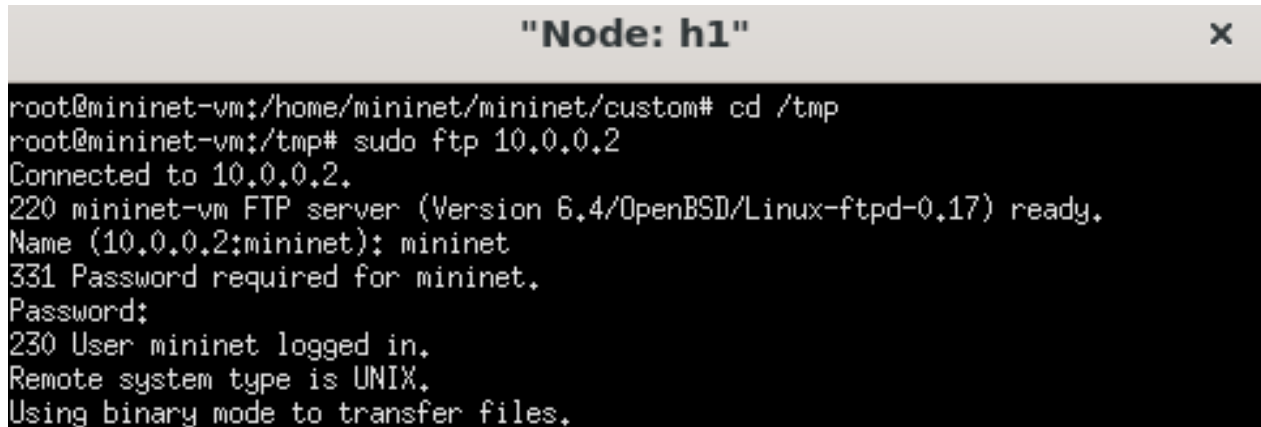
```

mininet@mininet-vm:~$ ls
Desktop  Downloads  mininet  oflops  openflow  pox  Templates
Documents  goku.png  Music    oftest  Pictures  Public  Videos
mininet@mininet-vm:~$ sudo tcpdump -w ftp-transfer host 10.0.0.1 and host 10.0.0.2 and port 21
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes

```

Figura 40: Captura TCPDUMP.

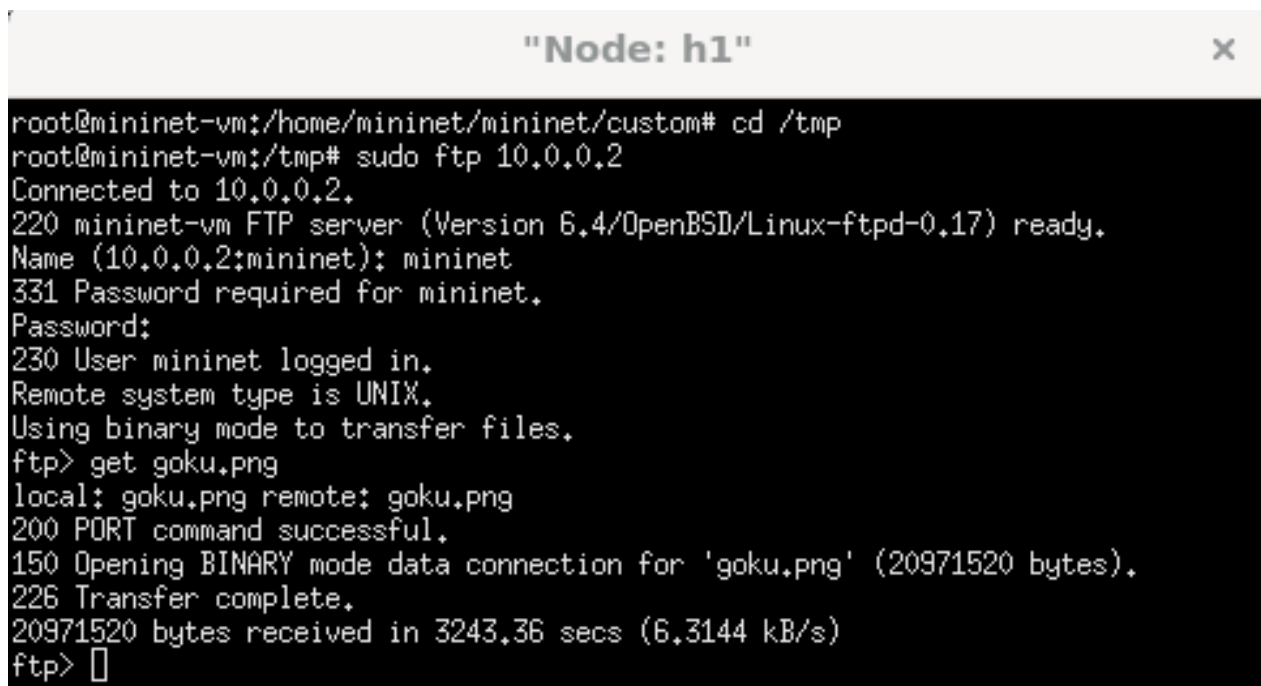
6. Inicio sesión FTP desde h1: Primeramente, el host de Chile(h1) se mueve de directorio a /tmp, para luego, iniciar una sesión FTP utilizando la dirección IP 10.0.0.2. Cabe destacar que para ingresar se debe usar Name:mininet y Password:mininet.



```
"Node: h1"
root@mininet-vm:/home/mininet/mininet/custom# cd /tmp
root@mininet-vm:/tmp# 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.
```

Figura 41: Inicio de sesión FTP desde host de Chile.

7. Obtener Archivo: Para este paso, a través de la sesión FTP, se procede a obtener el archivo goku.png a través del comando get. En este caso se pudo completar la transferencia en un tiempo de 3243,36 segundos, aproximadamente unos 55 minutos.



```
"Node: h1"
root@mininet-vm:/home/mininet/mininet/custom# cd /tmp
root@mininet-vm:/tmp# 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 goku.png
local: goku.png remote: goku.png
200 PORT command successful.
150 Opening BINARY mode data connection for 'goku.png' (20971520 bytes).
226 Transfer complete.
20971520 bytes received in 3243.36 secs (6.3144 kB/s)
ftp> □
```

Figura 42: Petición get en sesión FTP.

2.4.3. Captura y análisis de tráfico en Wireshark

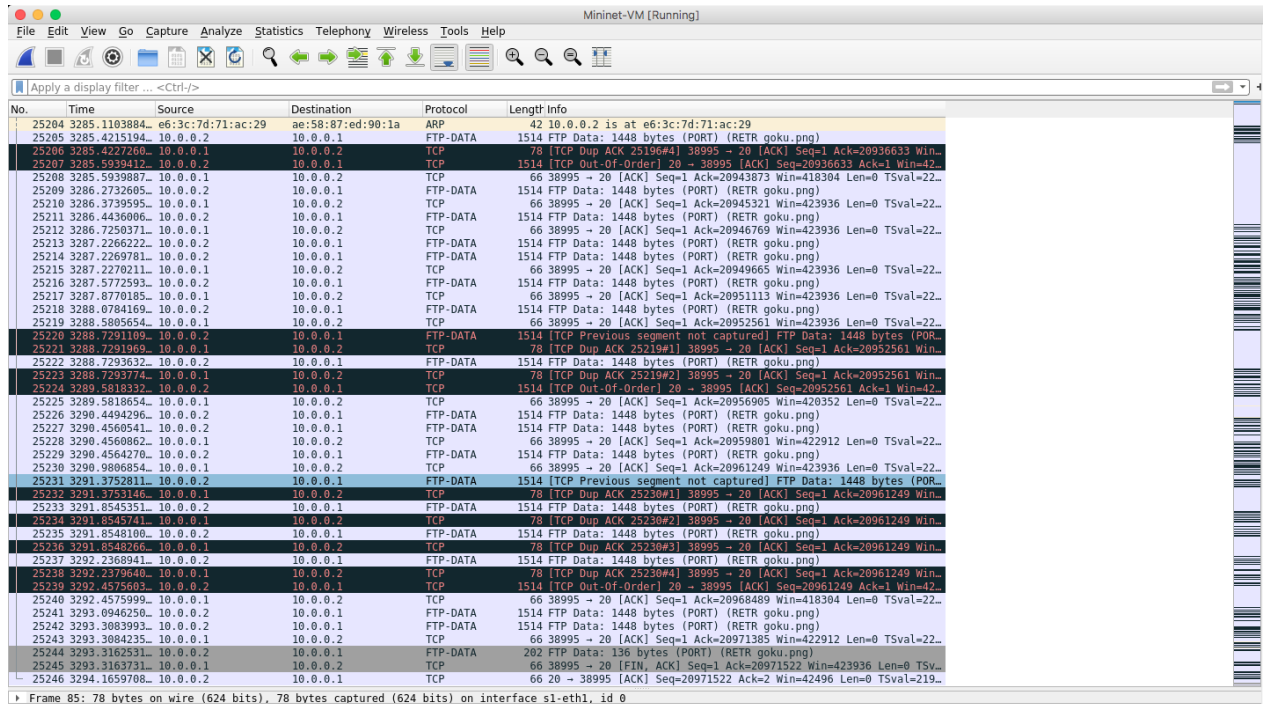
Primeramente, se captura el tráfico de red generado por la petición get del archivo goku.png hacia el servidor FTP del host de Australia.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.2	TCP	74	51062 → 21 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 T...
2	0.058957783	10.0.0.2	10.0.0.1	TCP	74	21 → 51062 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SA...
3	0.058957783	10.0.0.1	10.0.0.2	TCP	66	51062 → 21 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=2218965922...
4	1.713064344	10.0.0.2	10.0.0.1	FTP	138	Response: 220 mininet-vm FTP server (Version 6.4/OpenBSD/Linu...
5	1.713102251	10.0.0.1	10.0.0.2	TCP	66	51062 → 21 [ACK] Seq=1 Ack=73 Win=42496 Len=0 TSval=221896678...
6	6.096017613	e6:3c:7d:71:ac:29	ae:58:87:ed:90:1a	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
7	6.096043760	ae:58:87:ed:90:1a	e6:3c:7d:71:ac:29	ARP	42	10.0.0.1 is at ae:58:87:ed:90:1a
8	6.364600995	10.0.0.1	10.0.0.2	FTP	80	Request: USER mininet
9	9.220322112	10.0.0.2	10.0.0.1	TCP	66	21 → 51062 [ACK] Seq=73 Ack=15 Win=43520 Len=0 TSval=21949740...
10	9.221878482	10.0.0.2	10.0.0.1	FTP	102	Response: 331 Password required for mininet.
11	9.221896134	10.0.0.1	10.0.0.2	TCP	66	51062 → 21 [ACK] Seq=15 Ack=109 Win=42496 Len=0 TSval=2218974...
12	12.993904731	10.0.0.1	10.0.0.2	FTP	80	Request: PASS mininet
13	13.849428298	10.0.0.2	10.0.0.1	TCP	66	21 → 51062 [ACK] Seq=109 Ack=29 Win=43520 Len=0 TSval=2194978...
14	13.862183374	10.0.0.2	10.0.0.1	TCP	95	Response: 230 User mininet logged in.
15	13.862185599	10.0.0.1	10.0.0.2	TCP	66	51062 → 21 [ACK] Seq=29 Ack=138 Win=42496 Len=0 TSval=2218978...
16	13.862266951	10.0.0.1	10.0.0.2	FTP	72	Request: SYST
17	14.719845474	10.0.0.2	10.0.0.1	TCP	66	21 → 51062 [ACK] Seq=138 Ack=35 Win=43520 Len=0 TSval=2194979...
18	14.719963082	10.0.0.2	10.0.0.1	FTP	93	Response: 215 UNIX Type: L8 (Linux)
19	14.719982596	10.0.0.1	10.0.0.2	TCP	66	51062 → 21 [ACK] Seq=35 Ack=165 Win=42496 Len=0 TSval=2218979...
20	43.611792085	10.0.0.1	10.0.0.2	TCP	74	Request: TYPE I
21	45.540667679	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] 51062 → 21 [PSH, ACK] Seq=35 Ack=165 Win...
22	46.391677978	10.0.0.2	10.0.0.1	FTP	86	Response: 200 Type set to I.
23	46.391711818	10.0.0.1	10.0.0.2	TCP	66	51062 → 21 [ACK] Seq=43 Ack=185 Win=42496 Len=0 TSval=2219011...
24	46.391833582	10.0.0.1	10.0.0.2	FTP	88	Request: PORT 10,0,0,1,152,83
25	47.245811827	10.0.0.2	10.0.0.1	FTP	96	Response: 200 PORT command successful.
26	47.245842410	10.0.0.1	10.0.0.2	TCP	66	51062 → 21 [ACK] Seq=45 Ack=215 Win=42496 Len=0 TSval=2219012...
27	47.246129808	10.0.0.1	10.0.0.2	FTP	81	Request: RETR goku.png
28	48.140451976	10.0.0.2	10.0.0.1	TCP	66	21 → 51062 [ACK] Seq=215 Ack=80 Win=43520 Len=0 TSval=2195013...
29	102.6377707	10.0.0.1	10.0.0.1	TCP	74	20 → 38995 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 T...
30	102.6968660	10.0.0.1	10.0.0.2	TCP	74	38995 → 20 [SYN, ACK] Seq=0 Ack=1 Win=43440 Len=0 MSS=1460 SA...
31	49.95788474	10.0.0.2	10.0.0.1	TCP	66	20 → 38995 [ACK] Seq=1 Ack=1 Win=42496 Len=0 TSval=2195014815...
32	49.958034243	10.0.0.2	10.0.0.1	FTP	140	Response: 150 Opening BINARY mode data connection for 'goku.p...
33	49.961921622	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
34	49.961983963	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=1449 Win=42496 Len=0 TSval=2219015...
35	49.964295959	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
36	49.964310361	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=2897 Win=42496 Len=0 TSval=2219015...
37	49.964403739	10.0.0.2	10.0.0.1	FTP-DATA	1514	TCP Previous segment not captured] FTP Data: 1448 bytes (POR...
38	49.964479370	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 36#1] 38995 → 20 [ACK] Seq=1 Ack=2897 Win=42496...
39	49.964595367	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
40	49.964610345	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 36#2] 38995 → 20 [ACK] Seq=1 Ack=2897 Win=42496...

Figura 43: Captura Wireshark inicio transferencia FTP.

No.	Time	Source	Destination	Protocol	Length	Info
88	54.235842670	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
89	54.235871395	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=39097 Win=5683...
90	54.236018726	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
91	54.236035741	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=39097 Win=5990...
92	54.237951280	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
93	54.237966884	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=39097 Win=6297...
94	54.238008937	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
95	54.238103810	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=39097 Win=6553...
96	54.461339584	10.0.0.2	10.0.0.1	TCP	1514	[TCP Out-Of-Order] 20 → 38995 [ACK] Seq=39097 Ack=1 Win=42496...
97	54.461382864	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=49233 Win=68608 Len=0 TSval=221901...
98	55.088845244	10.0.0.1	10.0.0.1	FTP-DATA	1514	[TCP Previous segment not captured] FTP Data: 1448 bytes (POR...
99	55.088877460	10.0.0.2	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=49233 Win=7168...
100	55.091171448	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
101	55.091189209	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=49233 Win=7424...
102	55.314149281	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
103	55.314175800	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=49233 Win=7731...
104	55.943965750	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
105	55.943995082	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=49233 Win=8038...
106	55.945497166	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
107	55.945514240	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=49233 Win=8294...
108	56.161520845	10.0.0.2	10.0.0.1	TCP	1514	[TCP Out-Of-Order] 20 → 38995 [ACK] Seq=49233 Ack=1 Win=42496...
109	56.161564865	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=57921 Win=86016 Len=0 TSval=221902...
110	57.013366640	10.0.0.2	10.0.0.1	FTP-DATA	1514	[TCP Previous segment not captured] FTP Data: 1448 bytes (POR...
111	57.013397461	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=57921 Win=8857...
112	58.638902140	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
113	58.638934760	10.0.0.1	10.0.0.2	TCP	78	[TCP Window Update] 38995 → 20 [ACK] Seq=1 Ack=57921 Win=9164...
114	59.489359984	10.0.0.2	10.0.0.1	TCP	1514	[TCP Out-Of-Order] 20 → 38995 [ACK] Seq=57921 Ack=1 Win=42496...
115	59.489403631	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=62265 Win=94720 Len=0 TSval=221902...
116	59.489800653	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
117	59.489807369	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=63713 Win=97200 Len=0 TSval=221902...
118	60.338731414	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
119	60.338758473	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=65161 Win=97792 Len=0 TSval=221902...
120	60.341101610	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
121	60.384642814	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=66609 Win=97792 Len=0 TSval=221902...
122	61.189386150	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
123	61.191191829	10.0.0.1	10.0.0.2	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
124	61.191210428	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=69505 Win=97792 Len=0 TSval=221902...
125	61.232144342	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
126	61.301490897	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=70953 Win=97792 Len=0 TSval=221902...
127	62.045414280	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)

Figura 44: Captura Wireshark proceso transferencia FTP.



No.	Time	Source	Destination	Protocol	Length	Info
25204	3285.1183884	e6:3c:7d:71:ac:29	ae:58:87:ed:90:1a	ARP	42	10.0.0.2 is at e6:3c:7d:71:ac:29
25205	3285.4215194	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25206	3285.4227260	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 25196#4] 38995 → 20 [ACK] Seq=1 Ack=20936633 Win=...
25207	3285.5939412	10.0.0.2	10.0.0.1	TCP	1514	[TCP Out-Of-Order] 20 → 38995 [ACK] Seq=20936633 Ack=1 Win=42...
25208	3285.5939887	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20943873 Win=418304 Len=0 TSval=22...
25209	3286.2732685	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25210	3286.3735955	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20945321 Win=423936 Len=0 TSval=22...
25211	3286.4436066	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25212	3286.7250371	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20946769 Win=423936 Len=0 TSval=22...
25213	3287.2266222	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25214	3287.2269781	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25215	3287.2270211	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20949665 Win=423936 Len=0 TSval=22...
25216	3287.5772593	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25217	3287.8770185	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20951113 Win=423936 Len=0 TSval=22...
25218	3288.0784169	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25219	3288.5805654	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20952561 Win=423936 Len=0 TSval=22...
25220	3288.7291189	10.0.0.2	10.0.0.1	FTP-DATA	1514	[TCP Previous segment not captured] FTP Data: 1448 bytes (POR...
25221	3288.7291969	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 25219#1] 38995 → 20 [ACK] Seq=1 Ack=20952561 Win=...
25222	3288.7293632	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25223	3288.7293774	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 25219#2] 38995 → 20 [ACK] Seq=1 Ack=20952561 Win=...
25224	3289.5818332	10.0.0.2	10.0.0.1	TCP	1514	[TCP Out-Of-Order] 20 → 38995 [ACK] Seq=20952561 Ack=1 Win=42...
25225	3289.5818654	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20956905 Win=420352 Len=0 TSval=22...
25226	3290.4494296	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25227	3290.4509541	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25228	3290.4560862	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20959801 Win=422912 Len=0 TSval=22...
25229	3290.4564270	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25230	3290.9806854	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20961249 Win=423936 Len=0 TSval=22...
25231	3291.3752811	10.0.0.2	10.0.0.1	FTP-DATA	1514	[TCP Previous segment not captured] FTP Data: 1448 bytes (POR...
25232	3291.3753106	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 25230#1] 38995 → 20 [ACK] Seq=1 Ack=20961249 Win=...
25233	3291.8545351	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25234	3291.8545741	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 25230#2] 38995 → 20 [ACK] Seq=1 Ack=20961249 Win=...
25235	3291.8548100	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25236	3291.8548266	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 25230#3] 38995 → 20 [ACK] Seq=1 Ack=20961249 Win=...
25237	3292.2360941	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25238	3292.2370408	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 25230#4] 38995 → 20 [ACK] Seq=1 Ack=20961249 Win=...
25239	3292.4575803	10.0.0.2	10.0.0.1	TCP	1514	[TCP Out-Of-Order] 20 → 38995 [ACK] Seq=20961249 Ack=1 Win=42...
25240	3292.4575999	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20968489 Win=418304 Len=0 TSval=22...
25241	3293.0946250	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25242	3293.3083993	10.0.0.2	10.0.0.1	FTP-DATA	1514	FTP Data: 1448 bytes (PORT) (RETR goku.png)
25243	3293.3084235	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [ACK] Seq=1 Ack=20971385 Win=422912 Len=0 TSval=22...
25244	3293.3162531	10.0.0.2	10.0.0.1	FTP-DATA	202	FTP Data: 136 bytes (PORT) (RETR goku.png)
25245	3293.3163731	10.0.0.1	10.0.0.2	TCP	66	38995 → 20 [FIN, ACK] Seq=1 Ack=20971522 Win=423936 Len=0 TSv...
25246	3294.1659708	10.0.0.2	10.0.0.1	TCP	66	20 → 38995 [ACK] Seq=20971522 Ack=2 Win=42496 Len=0 TSval=219...

Figura 45: Captura Wireshark finalizada transferencia FTP.

Como se puede observar en la captura de tráfico de Wireshark, se observaron problemas relacionados con congestión de red, pérdida de paquetes y latencia, los cuales se producen debido a que hay mucho tráfico de red que se genera al transferir la imagen debido al tamaño y el diseño particular de la topología de red.

Como se puede observar en las capturas, estos problemas TCP lo resuelve a través de técnicas como el control de flujo, retransmisiones automáticas, control de congestión y ventana deslizante.

En este caso, no es recomendable utilizar un enlace UDP, debido a dos puntos:

- La red es propensa a la congestión y pérdida de paquetes debido al diseño de la topología de red y configuraciones como bw, delay y loss.
- UDP es protocolo que no garantiza la entrega de paquete debido a que no utiliza mecanismo de retransmisión y control de congestión.

Conclusiones y comentarios

En esta tarea se ha proporcionado una comprensión básica y práctica sobre el uso de la plataforma Mininet. A través de comandos básicos y la API de Python, se ha diseñado topologías personalizadas y las diversas configuraciones de red. También, gracias a la herramienta de Wireshark se analizó el tráfico de red y la transferencia de archivos mediante un

servicio FTP. Por otro lado, la simulación de una conexión entre Chile y Australia brindó una perspectiva practica de como se comportan las redes en situaciones realistas.

En conclusión, esta tarea ha establecido una base solida para futuras investigaciones y aplicaciones avanzadas en el campo de las redes definidas por software y en las arquitecturas emergentes.

3. GitHub

- <https://github.com/THELUXE1234/ArquiEmergentes>