

UNIVERSIDAD DIEGO PORTALES
FACULTAD DE INGENIERÍA Y CIENCIAS
ESCUELA DE INFORMÁTICA Y TELECOMUNICACIONES



Arquitecturas Emergentes
Tarea 01: Simulación de redes con Mininet

Profesor: Carlos García
Estudiante: Felipe Condore

Índice

1. Introducción	1
2. Configuración de Ambiente	1
3. Experimentos	2
3.1. Actividad 1	2
3.2. Actividad 02	5
3.3. Actividad 03	7
3.4. Actividad 4	9

1. Introducción

Uno de los aspectos más importantes en el último tiempo, corresponde a la virtualización de los diversos recursos TI, los cuales no solo corresponden a máquinas computacionales, sino también a diversos componentes que conforman la red.

Para el siguiente trabajo se lleva a cabo una serie de experimentos a través del Software de Mininet, el cual nos brinda la posibilidad de simular distintos entornos de una red, a través de un controlador virtual y global, para el manejo del tráfico.

Las pruebas realizadas pretender analizar el comportamiento de diversas acciones en la red, como lo son el establecimiento de un canal de TCP, comunicación mediante pings e intercambio de archivos a través de servicios de FTP, a través de redes cuyas topologías son diseñadas de manera personalizada con parámetros establecidos a través de código.

2. Configuración de Ambiente

Para lograr ejecutar la actividad, es necesario tener en consideración los siguientes requerimientos.

- Sistema Operativo: Ubuntu 18.04
- Docker 20.10.17

Para poder usar el software de mininet, va a ser necesario ocupar la imagen Containernet que se puede obtener de Docker Hub. Para lograr ciertos requerimientos es necesario crear una propia imagen, tanto de mininet como para los hosts con la finalidad de adquirir ciertas dependencias faltantes. A continuación se presentan los archivos Dockerfile con el cual generar las imágenes para cada equipo, junto a los comandos necesarios para llevar a cabo el despliegue del ambiente.

```
FROM containernet/containernet

RUN apt update

RUN apt install -y x11-xserver-utils

RUN apt install -y ifupdown net-tools

RUN apt install -y tcpdump

RUN apt install -y ftpd vim

RUN apt-get install -y xinetd telnetd telnet
```

Figura 1: Imagen de mininet en docker

```
FROM ubuntu:16.04
RUN apt-get update

RUN apt-get install -y libc6 libcap0.8 \
    apparmor libssl1.0.0 libssl-dev net-tools
RUN apt-get install -y traceroute iptables \
    arping ipcalc inetutils-ping
RUN apt-get install -y curl dnsutils wget vim \
    ethtool tcpdump \
    iperf \
    telnet \
    && apt clean
CMD ["bash"]
```

Figura 2: Imagen de los hosts en docker

```
# Descargar Imagen
docker pull containernet/containernet
# Instalar xhost
sudo apt install -y x11-xserver-utils
# Construir la imagen de mininet
docker build -t kunt-net ./mininet/
# Construir la imagen de host
docker build -t kunt-host ./host/
# Permisos de GUI
xhost +local:docker
# Despliegue del contenedor
sudo docker run --name containernet -it --rm --privileged --cap-add=SYS_MODULE --cap-add=SYS_NICE \
--cap-add=NET_ADMIN --network="host" --pid='host' -v /var/run/docker.sock:/var/run/docker.sock \
-v /lib/modules:/lib/modules -v /tmp/.X11-unix:/tmp/.X11-unix -v $(pwd)/codigos:/containernet/testing/ \
-e DISPLAY=$DISPLAY kunt-net /bin/bash
```

Figura 3: Comandos requeridos para el despliegue de Mininet en contenedores

Una vez dentro del contenedor, antes de realizar cualquiera de las pruebas anteriores, es necesario actualizar la lista de servidores DNS. Para ello pueden ejecutar el script de bash `dsn.sh` que se encuentra en el directorio `/containernet/testing/dsn.sh`. Este script va a agregar la ip de Google como un servidor DNS extra en la configuración de la máquina, para evitar problemas con la detección de nombres de sitios web públicos.

3. Experimentos

3.1. Actividad 1

Se lleva a cabo la implementación de la red mostrada en la figura 4, a través de un archivo de python. Para realizar esto, dentro del contenedor se utiliza el siguiente comando que permite desplegar la red a través de la topología personalizada.

```
#!/bin/bash
sudo mn --custom /containernet/testing/pregunta1.py \
--topo mytopo
```

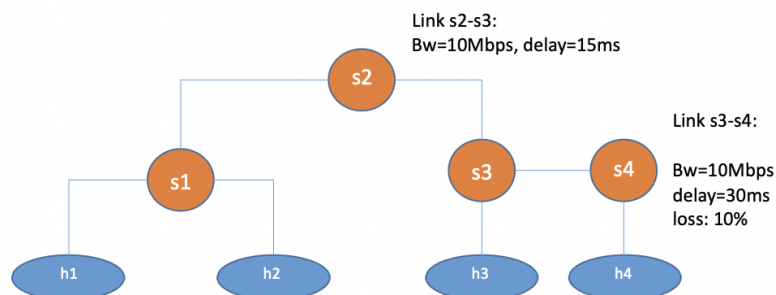


Figura 4: Topología de red - Actividad 01

Una vez dentro de la consola de containernet, es posible comprobar los nodos de la red (compuesta por los hosts y switches), a la vez que se puede obtener las direcciones ip asociadas a cada uno de los hosts, donde h1, h2, h3 y h4 poseen las direcciones 10.0.0.1, 10.0.0.2, 10.0.0.3 y 10.0.0.4 respectivamente.

```

containernet> h1 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:73 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:9615 (9.6 KB)  TX bytes:0 (0.0 B)

h1-eth0    Link encap:Ethernet  HWaddr 12:54:65:59:a6:9b
          inet addr:10.0.0.1  Bcast:0.0.0.0  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:85 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10568 (10.5 KB)  TX bytes:280 (280.0 B)

lo         Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

containernet>

```

(a) Configuración de red host h1

```

containernet> h2 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:03
          inet addr:172.17.0.3  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:74 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:9742 (9.7 KB)  TX bytes:0 (0.0 B)

h2-eth0    Link encap:Ethernet  HWaddr 9a:f1:ab:46:53:5e
          inet addr:10.0.0.2  Bcast:0.0.0.0  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:94 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11346 (11.3 KB)  TX bytes:280 (280.0 B)

lo         Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

containernet>

```

(b) Configuración de red host h2

```

containernet> h3 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:04
          inet addr:172.17.0.4  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:61 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:8036 (8.0 KB)  TX bytes:0 (0.0 B)

h3-eth0    Link encap:Ethernet  HWaddr 82:f1:1f:b0:dd:16
          inet addr:10.0.0.3  Bcast:0.0.0.0  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:97 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11660 (11.6 KB)  TX bytes:0 (0.0 B)

lo         Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

containernet>

```

(c) Configuración de red host h3

```

containernet> h4 ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:ac:11:00:05
          inet addr:172.17.0.5  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:51 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:6704 (6.7 KB)  TX bytes:0 (0.0 B)

h4-eth0    Link encap:Ethernet  HWaddr 0e:52:51:8a:88:0a
          inet addr:10.0.0.4  Bcast:0.0.0.0  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:96 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11590 (11.5 KB)  TX bytes:0 (0.0 B)

lo         Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

containernet>

```

(d) Configuración de red host h4

Figura 5: Configuración de redes para los host

Ingresamos a las terminales de cada host, a través de la opción de xterm en la consola de containernet (mininet). Una vez ingresado, corroboramos la conexión entre hosts mediante el uso de 10 pings entre h1 y h2, como en h1 y h3. Con estas muestras es posible determinar el delay correspondiente a los enlaces, puesto que la conexión de h1-h2 solamente atraviesa el switch s1 que no posee ningún tipo de delay asociado, donde se observa que el tiempo promedio de ida y vuelta es de 1,483 milisegundos. Mientras que la conexión entre h1-h3, atraviesa el enlace s2-s3 que posee un delay de 15 milisegundos, tomando en cuenta el viaje de ida y vuelta, se puede verificar que efectivamente se está realizando el retardo correspondiente, dado que se observa que el promedio de tiempo es de 30,797 milisegundos.

```

root@h1:~# ping -c 10 10.0.0.2
PING 10.0.0.2 (10.0.0.2): 56 data bytes
64 bytes from 10.0.0.2: icmp_seq=0 ttl=64 time=4.257 ms
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=8.331 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.908 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.267 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.222 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.181 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.179 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.181 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.171 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.182 ms
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.171/1.488/8.331/2.579 ms
root@h1:~#

```

(a) Ping realizado de h1 a h2

```

root@h1:~# ping -c 10 10.0.0.3
PING 10.0.0.3 (10.0.0.3): 56 data bytes
64 bytes from 10.0.0.3: icmp_seq=0 ttl=64 time=32.459 ms
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=30.657 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=30.577 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=30.629 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=30.617 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=30.636 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=30.666 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=30.590 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=30.648 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=30.491 ms
--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max/stddev = 30.491/30.797/32.459/0.556 ms
root@h1:~#

```

(b) Ping realizado de h1 a h3

Figura 6: Ejecución de pings

Para cada una de estas muestras, fue necesario iniciar una captura en wireshark con tal de corroborar el intercambio de paquetes a través de la red, tanto de ICMP como de paquetes ARP como se muestra en las imágenes a continuación.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=0/0, tt
2	0.000048	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=0/0, tt
3	1.000571	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=1/256,
4	0.000059	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=1/256,
5	0.994173	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=2/512,
6	0.000075	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=2/512,
7	1.000882	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=3/768,
8	0.000057	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=3/768,
9	1.001439	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=4/1024,
10	0.000053	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=4/1024,
11	1.001376	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=5/1280,
12	0.000043	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=5/1280,
13	0.221078	9a:f1:ab...	12:54:65...	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
14	0.008349	12:54:65...	9a:f1:ab...	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
15	0.000009	9a:f1:ab...	12:54:65...	ARP	42	10.0.0.2 is at 9a:f1:ab:46:53:5e
16	0.001303	12:54:65...	9a:f1:ab...	ARP	42	10.0.0.1 is at 12:54:65:59:a6:9b
17	0.770387	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=6/1536,
18	0.000042	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=6/1536,
19	1.001282	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=7/1792,
20	0.000043	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=7/1792,
21	1.001228	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=8/2048,
22	0.000041	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=8/2048,
23	1.001293	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0026, seq=9/2304,
24	0.000043	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0026, seq=9/2304,

Figura 7: Captura de Wireshark - Ping entre h1 y h2

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=0/0, tt
2	0.031547	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=0/0, tt
3	0.969144	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=1/256,
4	0.030435	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=1/256,
5	0.970589	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=2/512,
6	0.030333	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=2/512,
7	0.969895	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=3/768,
8	0.030408	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=3/768,
9	0.969962	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=4/1024,
10	0.030398	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=4/1024,
11	0.969858	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=5/1280,
12	0.030412	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=5/1280,
13	0.024601	82:f1:1f...	12:54:65...	ARP	42	Who has 10.0.0.1? Tell 10.0.0.3
14	0.000628	12:54:65...	82:f1:1f...	ARP	42	10.0.0.1 is at 12:54:65:59:a6:9b
15	0.944734	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=6/1536,
16	0.030406	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=6/1536,
17	0.969978	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=7/1792,
18	0.030365	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=7/1792,
19	0.970056	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=8/2048,
20	0.030403	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=8/2048,
21	0.970005	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x0029, seq=9/2304,
22	0.030273	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0029, seq=9/2304,

Figura 8: Captura de Wireshark - Ping entre h1 y h3

Una vez probada la conectividad de la red, se pone a prueba el enlace que conecta los switches s3 y s4 respectivamente, con el fin de lograr medir la configuración de retardos establecida a través del software. Es necesario comprobar el porcentaje de pérdidas del enlace establecido, para ello resulta necesario tomar una muestra lo suficientemente grande de pings entre los hosts h3 y h4, de tal forma de determinar un intervalo de confianza con 95 %.

Como se observa en la figura 9, se realiza un ping con una cantidad de 10000 request, cuyos resultados se describen en la siguiente tabla.

Mínimo	Promedio	Máximo	Desviación Estándar
60,121	60,632	153,851	1,047

Cuadro 1: hola

A partir de los datos del cuadro anterior, específicamente el promedio y la desviación estándar, es posible calcular el intervalo de confianza, el cual corresponde a 60,61 [ms] como límite inferior y 60,65 [ms] como límite superior.

3.2. Actividad 02

A través de la misma topología de red anterior, se va a implementar un servicio de http en el host h1, a la vez que el host h2 va a realizar una petición al servicio a través del comando wget. Para el caso del servidor se usa un módulo integrado de python, tanto cliente como servidor son llevados a cabo a través de la consola de containernet mediante las líneas de comando.

```
1 containernet> h1 python3 -m http.server 80 &
2 containernet> h2 wget -O - h1
```

El primer comando levantará el servidor en el equipo h1 de manera asíncrona a través del módulo http.server, habilitando el directorio actual donde se ejecuta (en este caso la carpeta raíz del sistema /) a través de html. Mientras que el segundo comando, wget se utiliza para descargar archivos a través de internet (http), particularmente la solicitud va dirigida a la ip del host h1 y la opción -O omite la descarga del archivo con la finalidad de desplegar su contenido en la propia terminal.

El tráfico de conexión entre ambos host logra visualizarse a través de wireshark. En la figura 10 a continuación se observan los paquetes filtrados por las ip's correspondiente a los hosts en la figura 11, donde se visualiza que el paquete N.º 4 preseleccionado posee el requerimiento GET en HTTP.

```
64 bytes from 10.0.0.4: icmp_seq=9976 ttl=64 time=60,820 ms
64 bytes from 10.0.0.4: icmp_seq=9978 ttl=64 time=60,627 ms
64 bytes from 10.0.0.4: icmp_seq=9979 ttl=64 time=60,643 ms
64 bytes from 10.0.0.4: icmp_seq=9980 ttl=64 time=60,670 ms
64 bytes from 10.0.0.4: icmp_seq=9981 ttl=64 time=60,766 ms
64 bytes from 10.0.0.4: icmp_seq=9982 ttl=64 time=60,817 ms
64 bytes from 10.0.0.4: icmp_seq=9984 ttl=64 time=60,770 ms
64 bytes from 10.0.0.4: icmp_seq=9985 ttl=64 time=60,640 ms
64 bytes from 10.0.0.4: icmp_seq=9986 ttl=64 time=60,587 ms
64 bytes from 10.0.0.4: icmp_seq=9987 ttl=64 time=60,603 ms
64 bytes from 10.0.0.4: icmp_seq=9988 ttl=64 time=60,625 ms
64 bytes from 10.0.0.4: icmp_seq=9989 ttl=64 time=60,667 ms
64 bytes from 10.0.0.4: icmp_seq=9990 ttl=64 time=60,724 ms
64 bytes from 10.0.0.4: icmp_seq=9992 ttl=64 time=60,821 ms
64 bytes from 10.0.0.4: icmp_seq=9993 ttl=64 time=60,581 ms
64 bytes from 10.0.0.4: icmp_seq=9994 ttl=64 time=60,575 ms
64 bytes from 10.0.0.4: icmp_seq=9995 ttl=64 time=60,626 ms
64 bytes from 10.0.0.4: icmp_seq=9996 ttl=64 time=60,689 ms
64 bytes from 10.0.0.4: icmp_seq=9998 ttl=64 time=60,828 ms
64 bytes from 10.0.0.4: icmp_seq=9999 ttl=64 time=60,685 ms
--- 10.0.0.4 ping statistics ---
10000 packets transmitted, 8092 packets received, 19% packet loss
round-trip min/avg/max/stddev = 60,121/60,632/153,851/1,047 ms
root@h3:~#
```

Figura 9: Resultados obtenidos del ping

```
containernet> h1 python3 -m http.server 80 &
containernet> h2 wget -O - h1
--2022-09-10 01:24:15-- http://10.0.0.1/
Connecting to 10.0.0.1:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 987 [text/html]
Saving to: 'STDOUT'

-          0%[          ] 0 --.-KB/s
/W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ascii">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="/.dockerenv">./dockerenv</a></li>
<li><a href="/bin">bin</a></li>
<li><a href="/boot">boot</a></li>
<li><a href="/dev">dev</a></li>
<li><a href="/etc">etc</a></li>
<li><a href="/home">home</a></li>
<li><a href="/lib">lib</a></li>
<li><a href="/lib64">lib64</a></li>
<li><a href="/media">media</a></li>
<li><a href="/mnt">mnt</a></li>
<li><a href="/opt">opt</a></li>
<li><a href="/proc">proc</a></li>
<li><a href="/root">root</a></li>
<li><a href="/run">run</a></li>
<li><a href="/sbin">sbin</a></li>
<li><a href="/srv">srv</a></li>
<li><a href="/sys">sys</a></li>
<li><a href="/tmp">tmp</a></li>
<li><a href="/usr">usr</a></li>
<li><a href="/var">var</a></li>
</ul>
<hr>
</body>
</html>
-          100%[=====] 987 --.-KB/s in 0s

2022-09-10 01:24:15 (82.9 MB/s) - written to stdout [987/987]

containernet> []
```

Figura 10: Solicitud mediante el comando wget

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.2	10.0.0.1	TCP	74	36538 → 80 [SYN] Seq=0 Win=42340
2	0.006415	10.0.0.1	10.0.0.2	TCP	74	80 → 36538 [SYN, ACK] Seq=0 Ack=1
3	0.000034	10.0.0.2	10.0.0.1	TCP	66	36538 → 80 [ACK] Seq=1 Ack=1 Win=4
4	0.000072	10.0.0.2	10.0.0.1	HTTP	201	GET / HTTP/1.1
5	0.000837	10.0.0.1	10.0.0.2	TCP	66	80 → 36538 [ACK] Seq=1 Ack=136 Win=4
6	0.000040	10.0.0.1	10.0.0.2	TCP	66	[TCP Dup ACK 5#1] 80 → 36538 [ACK]
7	0.003975	10.0.0.1	10.0.0.2	TCP	220	80 → 36538 [PSH, ACK] Seq=1 Ack=136 Win=4
8	0.000027	10.0.0.2	10.0.0.1	TCP	66	36538 → 80 [ACK] Seq=136 Ack=155 Win=4
9	0.000053	10.0.0.1	10.0.0.2	HTTP	1053	HTTP/1.0 200 OK (text/html)
10	0.000009	10.0.0.2	10.0.0.1	TCP	66	36538 → 80 [ACK] Seq=136 Ack=1142 Win=4
11	0.000074	10.0.0.1	10.0.0.2	TCP	66	80 → 36538 [FIN, ACK] Seq=1142 Ack=136
12	0.000566	10.0.0.2	10.0.0.1	TCP	66	36538 → 80 [FIN, ACK] Seq=136 Ack=1142
13	0.000024	10.0.0.1	10.0.0.2	TCP	66	80 → 36538 [ACK] Seq=1143 Ack=137 Win=4
14	5.009887	42:75:04:1b:8d:cb	36:1e:b3:c7:a4:5f	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
15	0.000712	36:1e:b3:c7:a4:5f	42:75:04:1b:8d:cb	ARP	42	10.0.0.1 is at 36:1e:b3:c7:a4:5f
16	0.012008	36:1e:b3:c7:a4:5f	42:75:04:1b:8d:cb	ARP	42	Who has 10.0.0.2? Tell 10.0.0.1
17	0.000040	42:75:04:1b:8d:cb	36:1e:b3:c7:a4:5f	ARP	42	10.0.0.2 is at 42:75:04:1b:8d:cb

Figura 11: Tráfico de red de solicitud GET a través de Wireshark

Cabe recalcar, que si se realiza nuevamente la prueba tomando en consideración la interfaz any, se logra observar que los equipos de la red envían más paquetes a través de la red. Tras filtrar y observar con mayor detalle el comportamiento del tráfico a través de las diversas interfaces, se puede concluir que el tráfico corresponde a la interfaz de Loopback. Esta interfaz tiene como funcionalidad establecer una conexión de red virtual, la cual muy posiblemente es utilizada por el controlador de containernet (mininet) para el uso de la propia red virtual.

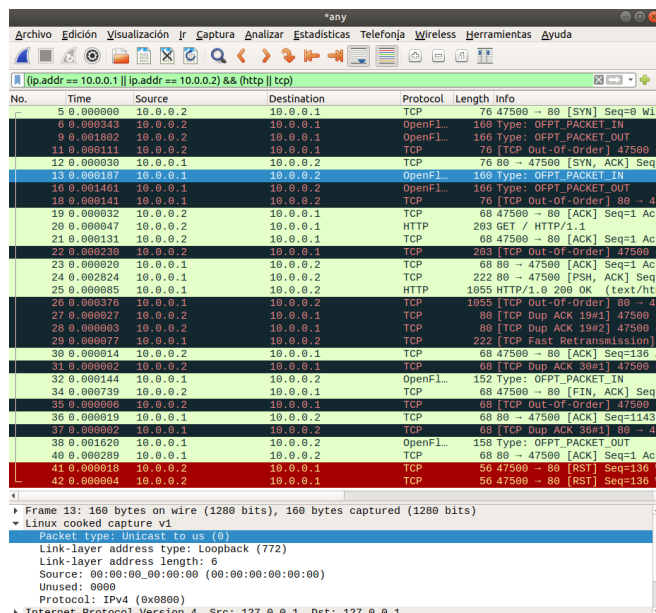


Figura 12: Solicitud mediante el comando wget

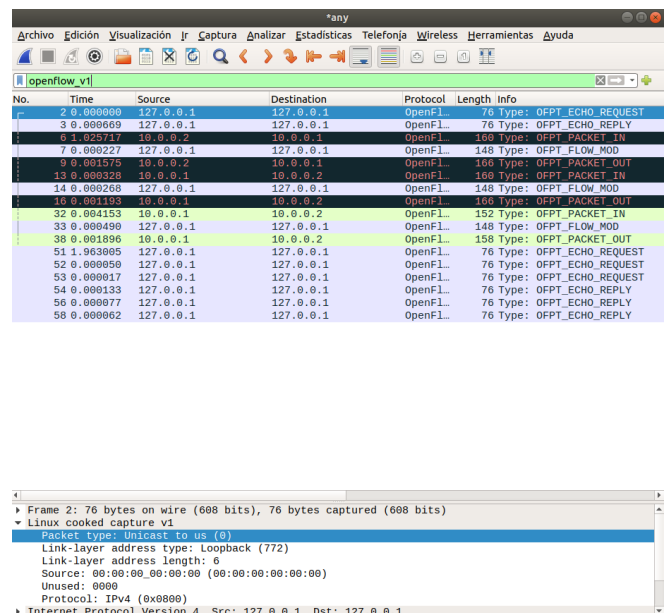


Figura 13: Solicitud mediante el comando wget

3.3. Actividad 03

La siguiente actividad consiste en la implementación de una topología de red tree mediante el protocolo NAT. Para realizar esto último, mininet nos brinda un archivo predeterminado ubicado en la siguiente ruta /container-net/examples/nat.py , el cual puede ser ejecutado con el siguiente comando dentro del contenedor.

```
1 #!/bin/bash
2 sudo python3 /container-net/examples/nat.py
```

Se comprueba que cada host de la red NAT tenga acceso a algún sitio público de internet como Facebook. Para corroborar esto último, se efectúa unos pings hacia el sitio web de Facebook en cada uno de los hosts, como se observa en las imágenes a continuación.

The figure shows four terminal windows, each representing a different node in the network (h1, h2, h3, h4). Each window displays the output of a ping command to the website www.facebook.com. The output includes the IP address of the destination (157.240.204.35), the size of the packet (64 bytes), the sequence number (icmp_seq), the time to live (ttl), and the round-trip time (time). The ping statistics at the bottom of each window show that 2 packets were transmitted and received with 0% packet loss and a time of 1001ms.

Figura 14: Solicitudes de ping a través de xterm

The figure shows a Wireshark capture of network traffic generated by pings to Facebook. The capture is filtered by 'dns || icmp'. The table below shows the first 53 packets of the capture.

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000000	10.0.0.1	8.8.8.8	DNS	87	Standard query 0x4289 A www.facebook.com OPT
4	0.000099	10.0.0.1	8.8.8.8	DNS	87	Standard query response 0x4289 A www.facebook.com OPT
5	0.005484	8.8.8.8	10.0.0.1	DNS	132	Standard query response 0x4289 A www.facebook.com CNAME star-mini.c10r.facebook.com A 157.240.204.35 OPT
6	0.002764	8.8.8.8	10.0.0.1	DNS	144	Standard query response 0xa0ad AAAA www.facebook.com CNAME star-mini.c10r.facebook.com AAAA 2a03:2880:f147:82:face:b00c:0:25de ...
7	0.005687	10.0.0.1	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0af, seq=1/256, ttl=64 (reply in 8)
8	0.006754	157.240.204.35	10.0.0.1	ICMP	98	Echo (ping) reply id=0xa0af, seq=1/256, ttl=52 (request in 7)
9	0.003696	10.0.0.1	8.8.8.8	DNS	98	Standard query 0xd4b3 PTR 35.204.240.157.in-addr.arpa OPT
10	0.009900	8.8.8.8	10.0.0.1	DNS	151	Standard query response 0xd4b3 PTR 35.204.240.157.in-addr.arpa PTR edge-star-mini-shv-01-scl2.facebook.com OPT
11	0.979083	10.0.0.1	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0af, seq=2/512, ttl=64 (reply in 12)
12	0.006384	157.240.204.35	10.0.0.1	ICMP	98	Echo (ping) reply id=0xa0af, seq=2/512, ttl=52 (request in 11)
13	1.422398	10.0.0.2	8.8.8.8	DNS	87	Standard query 0x5057 A www.facebook.com OPT
14	0.006374	10.0.0.2	8.8.8.8	DNS	87	Standard query 0xdc84 AAAA www.facebook.com OPT
15	0.001456	8.8.8.8	10.0.0.2	DNS	132	Standard query response 0x5057 A www.facebook.com CNAME star-mini.c10r.facebook.com A 157.240.204.35 OPT
16	0.005281	8.8.8.8	10.0.0.2	DNS	144	Standard query response 0xdc84 AAAA www.facebook.com CNAME star-mini.c10r.facebook.com AAAA 2a03:2880:f147:82:face:b00c:0:25de ...
17	0.004259	10.0.0.2	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0a5, seq=1/256, ttl=64 (reply in 18)
18	0.009164	157.240.204.35	10.0.0.2	ICMP	98	Echo (ping) reply id=0xa0a5, seq=1/256, ttl=52 (request in 17)
19	0.004475	10.0.0.2	8.8.8.8	DNS	98	Standard query 0xdcd1 PTR 35.204.240.157.in-addr.arpa OPT
20	0.000951	8.8.8.8	10.0.0.2	DNS	151	Standard query response 0xdcd1 PTR 35.204.240.157.in-addr.arpa PTR edge-star-mini-shv-01-scl2.facebook.com OPT
21	0.977827	10.0.0.2	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0a5, seq=2/512, ttl=64 (reply in 22)
22	0.007707	157.240.204.35	10.0.0.2	ICMP	98	Echo (ping) reply id=0xa0a5, seq=2/512, ttl=52 (request in 21)
27	1.786860	10.0.0.3	8.8.8.8	DNS	87	Standard query 0x48d2 AAAA www.facebook.com OPT
28	0.006395	10.0.0.3	8.8.8.8	DNS	87	Standard query 0x909e A www.facebook.com OPT
30	1.392422	8.8.8.8	10.0.0.3	DNS	144	Standard query response 0x48d2 AAAA www.facebook.com CNAME star-mini.c10r.facebook.com AAAA 2a03:2880:f147:82:face:b00c:0:25de ...
31	0.009036	8.8.8.8	10.0.0.3	DNS	132	Standard query response 0x909e A www.facebook.com CNAME star-mini.c10r.facebook.com A 157.240.204.35 OPT
32	0.014839	10.0.0.3	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0a7, seq=1/256, ttl=64 (reply in 33)
33	0.006749	157.240.204.35	10.0.0.3	ICMP	98	Echo (ping) reply id=0xa0a7, seq=1/256, ttl=52 (request in 32)
34	0.014283	10.0.0.3	8.8.8.8	DNS	98	Standard query 0xa409 PTR 35.204.240.157.in-addr.arpa OPT
35	0.006386	8.8.8.8	10.0.0.3	DNS	151	Standard query response 0xa409 PTR 35.204.240.157.in-addr.arpa PTR edge-star-mini-shv-01-scl2.facebook.com OPT
38	0.973759	10.0.0.3	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0a7, seq=2/512, ttl=64 (reply in 39)
39	0.006945	157.240.204.35	10.0.0.3	ICMP	98	Echo (ping) reply id=0xa0a7, seq=2/512, ttl=52 (request in 38)
42	1.597883	10.0.0.4	8.8.8.8	DNS	87	Standard query 0x3a81 A www.facebook.com OPT
43	0.002612	10.0.0.4	8.8.8.8	DNS	87	Standard query 0xc498 AAAA www.facebook.com OPT
44	0.004695	8.8.8.8	10.0.0.4	DNS	132	Standard query response 0x3a81 A www.facebook.com CNAME star-mini.c10r.facebook.com A 157.240.204.35 OPT
45	0.021314	8.8.8.8	10.0.0.4	DNS	144	Standard query response 0xc498 AAAA www.facebook.com CNAME star-mini.c10r.facebook.com AAAA 2a03:2880:f147:82:face:b00c:0:25de ...
46	0.002161	10.0.0.4	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0a1, seq=1/256, ttl=64 (reply in 47)
47	0.008022	157.240.204.35	10.0.0.4	ICMP	98	Echo (ping) reply id=0xa0a1, seq=1/256, ttl=52 (request in 46)
48	0.003106	10.0.0.4	8.8.8.8	DNS	98	Standard query 0x5273 PTR 35.204.240.157.in-addr.arpa OPT
49	0.011564	8.8.8.8	10.0.0.4	DNS	151	Standard query response 0x5273 PTR 35.204.240.157.in-addr.arpa PTR edge-star-mini-shv-01-scl2.facebook.com OPT
52	0.970907	10.0.0.4	157.240.204.35	ICMP	98	Echo (ping) request id=0xa0a1, seq=2/512, ttl=64 (reply in 53)
53	0.005758	157.240.204.35	10.0.0.4	ICMP	98	Echo (ping) reply id=0xa0a1, seq=2/512, ttl=52 (request in 52)

Figura 15: Tráfico de red generado por los pings a Facebook

Finalmente, montamos el mismo servicio de python usado en la actividad anterior, accediendo al sitio web del host utilizado a través de la IP del mismo, como se observa en la imagen a continuación.

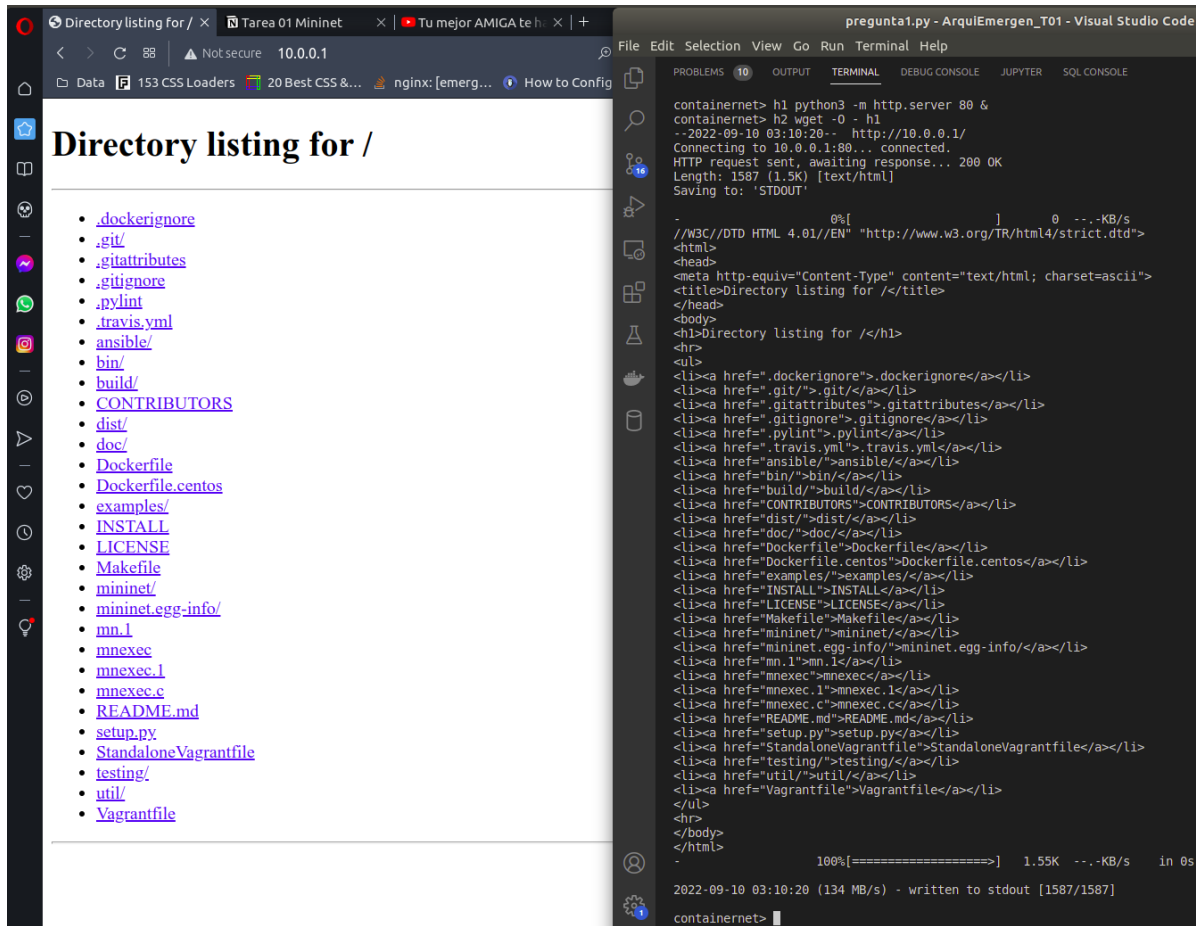


Figura 16: Acceso al servicio http del host h1 a través de navegador Opera en la máquina local

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.5	224.0.0.251	MDNS	183	Standard query 0x0000 PTR
2	0.280858	fe80::c4af:e7ff:fee...	ff02::2	ICMPv6	70	Router Solicitation from c
3	1.382359	fe80::c4af:e7ff:fee...	ff02::fb	MDNS	203	Standard query 0x0000 PTR
4	3.540357	aa:d8:10:b4:bb:ca	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.
5	9.210304	2e:1e:9b:93:7d:9e	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.
6	0.004471	ba:19:3b:c4:b8:75	2e:1e:9b:93:7d:9e	ARP	42	10.0.0.1 is at ba:19:3b:c4
7	0.000007	10.0.0.5	10.0.0.1	TCP	74	54180 → 80 [SYN] Seq=0 Win
8	0.003282	10.0.0.1	10.0.0.5	TCP	74	80 → 54180 [SYN, ACK] Seq=
9	0.000030	10.0.0.5	10.0.0.1	TCP	66	54180 → 80 [ACK] Seq=1 Ack
10	0.008728	10.0.0.5	10.0.0.1	TCP	74	54182 → 80 [SYN] Seq=0 Win
11	0.002294	10.0.0.5	10.0.0.1	HTTP	527	GET / HTTP/1.1
12	0.000165	10.0.0.1	10.0.0.5	TCP	66	80 → 54180 [ACK] Seq=1 Ack
13	0.001172	10.0.0.1	10.0.0.5	TCP	221	80 → 54180 [PSH, ACK] Seq=
14	0.000015	10.0.0.5	10.0.0.1	TCP	66	54180 → 80 [ACK] Seq=462 A
15	0.000095	10.0.0.1	10.0.0.5	HTTP	1653	HTTP/1.0 200 OK (text/htm
16	0.000007	10.0.0.5	10.0.0.1	TCP	66	54180 → 80 [ACK] Seq=462 A
17	0.000059	10.0.0.1	10.0.0.5	TCP	66	80 → 54180 [FIN, ACK] Seq=
18	0.001189	10.0.0.1	10.0.0.5	TCP	74	80 → 54182 [SYN, ACK] Seq=
19	0.000019	10.0.0.5	10.0.0.1	TCP	66	54182 → 80 [ACK] Seq=1 Ack
20	0.000696	10.0.0.5	10.0.0.1	TCP	66	54180 → 80 [FIN, ACK] Seq=
21	0.000025	10.0.0.1	10.0.0.5	TCP	66	80 → 54180 [ACK] Seq=1744
22	5.068895	ba:19:3b:c4:b8:75	2e:1e:9b:93:7d:9e	ARP	42	Who has 10.0.0.5? Tell 10.
23	0.000009	2e:1e:9b:93:7d:9e	ba:19:3b:c4:b8:75	ARP	42	10.0.0.5 is at 2e:1e:9b:93

Figura 17: Captura de tráfico de ingreso por navegador

Como se logra observar en la captura de wireshark, las ip de los hosts involucrados corresponden a 10.0.0.1 (host donde se encuentra el servicio http) y 10.0.0.5 (correspondiente al NAT), dado que la configuración de red NAT habilita la asignación de direcciones tanto internas como externas, es normal que la máquina local en vez de comunicarse con el servicio con su propia ip, lo haga a través de la dirección de NAT en la topología virtual. Considerando que la captura es realizada a través de la interfaz NAT dispuesta en la red, es normal que en la muestra de tráfico no se vea reflejado los paquetes de solicitud realizados por el host h2 a través de la terminal de containernet mostrado en la figura 16 y solamente se vea las solicitudes realizadas desde el exterior de la red, como lo es el navegador de Opera.

3.4. Actividad 4

Para esta actividad se implementa una topología diferente a la anterior, correspondiente a la figura 18. Se analiza el enlace entre los hosts de Chile y Australia mediante el comando iperf, el cual es una herramienta de red de código abierto utilizada para medir el rendimiento de una red. Se puede utilizar para probar el funcionamiento de canales a través de TCP y UDP.

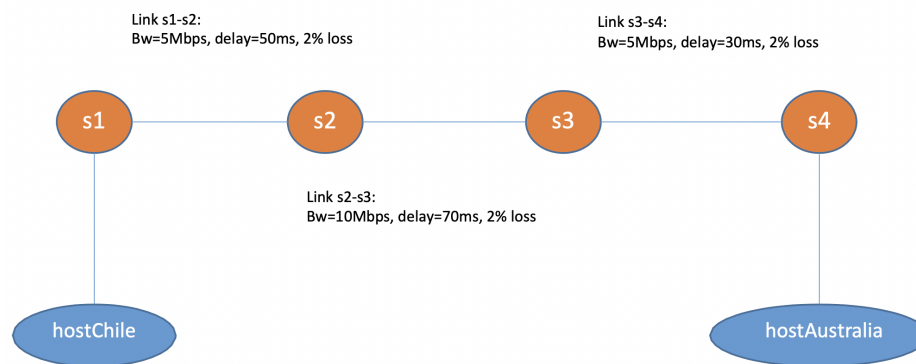


Figura 18: Topología de red - Actividad 04

Como se observa en la figura 19 y 20, el comando iperf los brinda tanto los intervalos, la cantidad de datos transferidos y el ancho de banda del canal de transporte TCP. A través de estos, es posible medir el impacto que tiene las pérdidas en los enlaces entre los hosts de Chile y Australia, aún considerando que la red corresponde a una topología serial, el ancho de banda de menor alcance corresponde a 5 Megabytes por segundo. Pero como se logra observar en los resultados, tanto la pérdida como el delay generan que la conexión establecida posea un ancho de banda alrededor de 240 Kilobytes por segundo aproximadamente.

```
containernet> iperf h1 h2
*** Iperf: testing TCP bandwidth between h1 and h2
*** Results: ['221 Kbits/sec', '266 Kbits/sec']
containernet> []
```

Figura 19: Ejecución de iperf a través de containernet

```
root@h1:/
Reading package lists... Done
root@h1:/# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85,3 KByte (default)
-----
[  4] local 10.0.0.1 port 5001 connected with 10.0.0.2 port 34286
[ ID] Interval      Transfer    Bandwidth
[  4] 0.0-17.6 sec   512 KBytes  238 Kbits/sec
root@h1:/#
```

```
root@h2:/
root@h2:/# iperf -c 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85,3 KByte (default)
-----
[  3] local 10.0.0.2 port 34286 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[  3] 0.0-15.8 sec   512 KBytes  266 Kbits/sec
root@h2:/#
```

Figura 20: Ejecución de iperf cliente/servidor

Una vez caracterizado el enlace, se procede a realizar una descarga de un archivo entre hosts a través de la red. Para realizar esto, ambos host se comunicarán bajo una arquitectura de cliente/servidor. El equipo h2 va a contener un servidor FTP, el cual brindará acceso remoto sobre las carpetas de usuarios creados dentro de la máquina. Mientras que el equipo h1, instalará un cliente ftp para acceder a las carpetas de usuarios brindados por el servidor FTP remoto, con la finalidad de descargar algún archivo disponible y examinar de manera analítica el tráfico generado entre el host h1 y el switch s1.

Servidor

Se crea a un usuario dentro de la máquina con nombre de usuario y contraseña admin, generando su carpeta correspondiente en el directorio /home . Luego, en la carpeta de este usuario, se genera una imagen en formato png de 10 Mb de tamaño a través del comando fallocation, como se observa en la figura 21.

Luego instalamos e iniciamos el servicio de FTP a través de las siguientes líneas de comando. Donde también se verifica que el servicio se encuentre operativo a través del puerto 21, como se muestra en la figura 22.

```

root@h2: /home/admin
root@h2:~# ls
bin boot dev etc home lib lib64 media mnt opt
root@h2:~# cd home/
root@h2:~/home# ls
root@h2:~/home# useradd --create-home admin
root@h2:~/home# passwd admin
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@h2:~/home# ls
admin
root@h2:~/home# cd admin/
root@h2:~/home/admin# fallocation -l 10MiB un-archivo.png
root@h2:~/home/admin# ls
un-archivo.png
root@h2:~/home/admin#

```

Figura 21: Creación de usuarios y archivo

```

1 # Instalación
2 apt update
3 apt install -y ftpd
4
5 # Despliegue
6 inetd
7
8 # Verificación - Puerto 21
9 netstat -tulnp

```

```

root@h2:~/home/admin# inetd
root@h2:~/home/admin# netstat -tulnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address   Foreign Address State     PID/Program name
tcp        0      0 0.0.0.0:0.0.0.0:21 0.0.0.0:*      LISTEN    606/inetd
root@h2:~/home/admin#

```

Figura 22: Verificación del servicio ftp - Port 21

Cliente

Ingresamos al directorio /tmp donde se iniciará la descarga del documento a través de consola. Luego se instala un cliente FTP y se conecta al servicio iniciado por el host h2 a través de su ip 10.0.0.2 . Una vez abierto el cliente FTP, nos autenticamos con las credenciales del usuario admin creado previamente en el servidor. Finalmente, cuando se establezca la conexión, ejecutamos el comando GET un-archivo.png, este iniciará la descarga del documento alojado en la carpeta del usuario admin proveniente del servidor, el cual fue creado pasos atrás. Los pasos descritos se pueden observar en la figura 23, al igual que la captura de paquetes a través de Wireshark mostrada en la figura 24.

```

root@h1:~# cd /tmp
root@h1:~/tmp# ls
root@h1:~/tmp# ftp 10.0.0.2
Connected to 10.0.0.2.
220 h2 FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (10.0.0.2:root): admin
331 Password required for admin.
Password:
230 User admin logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for '/bin/ls'.
total 10256
-rw-r--r-- 1 admin admin    16 Sep 10 05:48 .bash_history
-rw-r--r-- 1 admin admin   220 Aug 31 2015 .bash_logout
-rw-r--r-- 1 admin admin  3771 Aug 31 2015 .bashrc
-rw-r--r-- 1 admin admin    655 May 16 2017 .profile
-rw-r--r-- 1 root root 10485760 Sep 10 05:31 un-archivo.png
226 Transfer complete.
ftp> get un-archivo.png
local: un-archivo.png remote: un-archivo.png
200 PORT command successful.
150 Opening BINARY mode data connection for 'un-archivo.png' (10485760 bytes).
226 Transfer complete.
10485760 bytes received in 655.97 secs (15.6106 kB/s)
ftp> 221 Goodbye.
root@h1:~/tmp# ls
un-archivo.png
root@h1:~/tmp#

```

Figura 23: Descarga del archivo

No.	Time	Source	Destination	Protocol	Length	Info
6488	0.008537	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6489	0.018434	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6490	0.026137	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5208457 Win=431184 Len=0 TSval=2264405638 TSecr=4140135546
6491	0.028118	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5211353 Win=436224 Len=0 TSval=2264405666 TSecr=4140135546
6492	0.267812	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6493	0.030786	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6494	0.025858	10.0.0.2	10.0.0.1	FTP-DA...	1514	[TCP Previous segment not captured] FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6495	0.041582	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5214249 Win=436224 Len=0 TSval=2264406026 TSecr=4140135892
6496	0.020806	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 6495#1] 47373 → 20 [ACK] Seq=1 Ack=5214249 Win=436224 Len=0 TSval=2264406049 TSecr=4140135892
6497	0.284363	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6498	0.005740	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6499	0.020245	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6500	0.043258	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 6495#2] 47373 → 20 [ACK] Seq=1 Ack=5214249 Win=436224 Len=0 TSval=2264406395 TSecr=4140136576
6501	0.004585	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 6495#3] 47373 → 20 [ACK] Seq=1 Ack=5214249 Win=436224 Len=0 TSval=2264406402 TSecr=4140136576
6502	0.016951	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 6495#4] 47373 → 20 [ACK] Seq=1 Ack=5214249 Win=436224 Len=0 TSval=2264406422 TSecr=4140136576
6503	0.183705	10.0.0.2	10.0.0.1	TCP	1514	[TCP Out-Of-Order] 20 → 47373 [ACK] Seq=5214249 Ack=1 Win=43088 Len=1448 TSval=4140136576 TSecr=4140136576
6504	0.054100	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5221489 Win=432128 Len=0 TSval=2264406666 TSecr=4140136576
6505	0.053815	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6506	0.118144	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5222937 Win=437248 Len=0 TSval=2264406835 TSecr=4140136683
6507	0.119294	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6508	0.166251	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5224385 Win=437248 Len=0 TSval=2264407123 TSecr=4140136924
6509	0.022759	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6510	0.011912	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6511	0.061616	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5227281 Win=437248 Len=0 TSval=2264407216 TSecr=4140137091
6512	0.197046	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6513	0.102679	10.0.0.2	10.0.0.1	FTP-DA...	1514	FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6514	0.065986	10.0.0.1	10.0.0.2	TCP	66	47373 → 20 [ACK] Seq=1 Ack=5230177 Win=437248 Len=0 TSval=2264407579 TSecr=4140137373
6515	0.285494	10.0.0.2	10.0.0.1	FTP-DA...	1514	[TCP Previous segment not captured] FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6516	0.065774	10.0.0.1	10.0.0.2	TCP	78	[TCP Dup ACK 6514#1] 47373 → 20 [ACK] Seq=1 Ack=5230177 Win=437248 Len=0 TSval=2264407930 TSecr=4140137373
6517	0.214882	10.0.0.2	10.0.0.1	FTP-DA...	1514	[TCP Previous segment not captured] FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6518	0.066396	10.0.0.1	10.0.0.2	TCP	86	[TCP Dup ACK 6514#2] 47373 → 20 [ACK] Seq=1 Ack=5230177 Win=437248 Len=0 TSval=2264408212 TSecr=4140137373
6519	0.008323	10.0.0.2	10.0.0.1	FTP-DA...	1514	[TCP Fast Retransmission] FTP Data: 1448 bytes (PORT) (RETR un-archivo.png)
6520	0.068140	10.0.0.1	10.0.0.2	TCP	78	47373 → 20 [ACK] Seq=1 Ack=5233073 Win=435200 Len=0 TSval=2264408287 TSecr=4140138191 SLE=5234521

Figura 24: Tráfico FTP obtenido por la descarga del archivo

Análisis

¿Qué problemas se puede observar en la transmisión TCP de paquetes? ¿Por qué se producen? ¿Cómo resuelve TCP este problema?

En el tráfico medido a través de Wireshark, como se observa en la figura 24, existe una gran cantidad de paquetes perdidos o retrasos en la transmisión de ciertos segmentos, lo que resulta crucial para el protocolo de transporte de TCP, puesto que dentro de sus objetivos también es necesario entregar la información en un orden establecido, lo que provoca una mayor demora en el tiempo total para completar la descarga del archivo en cuestión.

La principal causa es la propia inestabilidad del canal, más específicamente las de los enlaces entre los switch, dado que el alto delay junto al bajo ancho de banda de cada uno de estos, incrementa acumulativamente el riesgo de que ocurra alguna pérdida o retraso en la transmisión, volviendo al canal un medio en extremo inestable y poco predecible. Por estos motivos se observa una gran cantidad de fallas en la comunicación entre el cliente y el servidor.

Para resolver esta problemática, el propio protocolo TCP tiene ciertas políticas, como la retransmisión de paquetes o duplicaciones de ACK entre muchas otras, para mantener así una cohesión de la información transmitida, a través de nuevos tipos de mensajes o flags que mantienen a ambos equipos coordinados en la manera y orden de transmisión de datos, aunque estos mismos mecanismos son los que provocan que se demora en gran medida el tiempo de descarga, llegando a demorar alrededor de 6 minutos con 55 segundos.

¿Sería conveniente usar un enlace UDP para la transferencia de archivos en este tipo de enlace?

A diferencia de TCP, el protocolo UDP no se preocupa en mantener una conexión o transmisión ordenadas de los datos, al igual que tampoco coordina entre ambos equipos el correcto recibimiento de los segmentos enviados. Por esto mismo, el protocolo UDP es recomendable usarlo en contexto donde se requiera una transmisión en tiempo real de datos, como pueden ser distintas aplicaciones de streaming o big data y no para transferencia de archivos, puesto que no posee estas cualidades para verificar el correcto envío de información que si posee TCP.

Aun así, cabe recalcar que el protocolo FTP al proveer estos mecanismos genera un problema aún más grande, debido a que incrementa considerablemente la congestión de la propia red a causa de la inestabilidad de los enlaces. Las consecuencias de este canal tan débil sumado a las políticas del protocolo se pueden ver reflejadas en la información especializada de Wireshark mostrada en la figura 25, donde más de 5000 paquetes (cerca del 40 % de la muestra total) corresponden a errores de transmisión o corrección por TCP, lo cual incrementa en gran medida la latencia y congestión de la red.

Finalmente, se puede concluir que si bien TCP está diseñado para la transferencia correcta de archivos, no cumple o satisface todas las necesidades para la arquitectura de red que se está simulando. Debido a esto, y la necesidad de una red con el tipo de enlaces con baja estabilidad, se puede determinar que para estas situaciones resulta mucho más conveniente utilizar el protocolo UDP, para realizar este tipo transferencias con enlaces de estas características.

Gravedad	Resumen	Grupo	Protocolo	Recuento
> Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	691
> Warning	D-SACK Sequence	Sequence	TCP	2597
> Warning	Previous segment(s) not captured (common at capture st...	Sequence	TCP	733
> Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	8
> Note	This frame is a (suspected) fast retransmission	Sequence	TCP	67
> Note	This frame is a (suspected) retransmission	Sequence	TCP	100
> Note	Duplicate ACK (#1)	Sequence	TCP	2164
> Chat	Connection finish (FIN)	Sequence	TCP	2
> Chat	TCP window update	Sequence	TCP	223
> Chat	Connection establish acknowledge (SYN+ACK): server por...	Sequence	TCP	2
> Chat	Connection establish request (SYN): server port 21	Sequence	TCP	2

Figura 25: Información especializada de Wireshark