

Trabajo Práctico 3

[75.43] Introducción a los sistemas distribuidos
Primer cuatrimestre de 2022

Fabbiano, Fernando	102464	ffabbiano@fi.uba.ar
Nocetti, Tomas	100853	tnocetti@fi.uba.ar
Alasino, Franco	102165	falasino@fi.uba.ar
Sportelli Castro, Luciano	99565	lsportelli@fi.uba.ar
Ganopolsky, Damian	101168	dganopolsky@fi.uba.ar

Índice

1. Introducción	2
2. Hipótesis y soluciones realizadas	2
3. Implementación	2
3.1. Topología	2
3.2. Firewall	2
4. Pruebas	3
4.1. 1: Pingall Wireshark desde H1	4
4.2. 2: Bloqueo de puerto 80 H1-H2	4
4.3. 3: Bloqueo de paquetes del H1 con puerto 5001 y protocolo UDP	5
4.4. 4: Bloqueo de paquetes entre H1 y H4	7
5. Preguntas a responder	8
6. Dificultades encontradas	9
6.1. Primer encuentro con las herramientas	9
6.2. UDP	9
7. Conclusiones	9

1. Introducción

El objetivo final del trabajo es implementar una topología de red utilizando la librería 'mininet' y sobre esta topología definir un conjunto de reglas, funcionando como un Firewall.

La primera parte del trabajo consiste en poder armar la topología correctamente, para poder verificar su funcionamiento se utilizara el comando 'pingall', y de funcionar bien, se debería poder ver que la señal haya llegado a todos los hosts, desde todos los hosts.

Luego, se modificara el controlador, para que funcione como un Firewall, agregándole 3 reglas. Para el controlador OpenFlow se hará uso de la API de Python POX, luego para verificar el correcto funcionamiento de las reglas se utilizara IPerf. IPerf nos servira para saber si los paquetes que se envian llegan al destino o quedan bloqueados en el envío por el 'Firewall'.

2. Hipótesis y soluciones realizadas

Para el trabajo se tomaron las siguientes hipótesis

- Para la regla "Se debe elegir dos hosts cualquiera, y los mismos no deben poder comunicarse de ninguna forma" bloqueamos cualquier tipo de comunicación que se haga por medio de la red IP.
- Filtrar un paquete será enviar dicho paquete al puerto 'OFPP-NONE' que es una constante provista por pox que hará que el paquete no vaya a ningún lado

3. Implementación

3.1. Topología

Para la implementación de la topología indicada en el enunciado creamos una clase que en su constructor agrega los 4 hosts fijos y luego mediante un bucle agrega n (parámetro) switches y también los links correspondientes, conectándolos para formar la topología lineal pedida en el enunciado. Para esta primera parte utilizamos el simulador de redes mininet.

3.2. Firewall

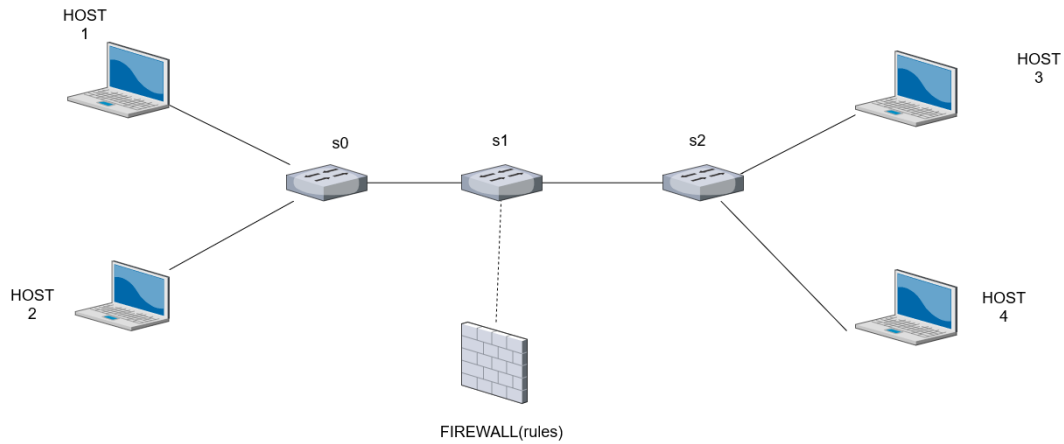
Para la segunda parte del trabajo utilizamos el controlador POX. Para esto creamos una clase 'controller.py' que establece las reglas en el switch configurado como firewall según el archivo de configuración 'config.py', a los demás switches no les establecerá las reglas. Es suficiente igualmente que en un switch se satisfagan las reglas para influir a toda la topología, ya que, esta misma es lineal y al dropearse los paquetes en uno de los switches, no llegaran los paquetes que se quieran enviar al otro extremo.

Para filtrar el paquete lo que hicimos fue establecer como destinatario del paquete un puerto no valido, para esto utilizamos la constante 'OFPP-NONE' que hace que la salida no vaya a ningún lado, esto lo leímos en la documentación de pox. Cuando se cumple alguna de las reglas, se dropea el paquete.

Para la primer regla, establecimos 2 posibles matches, uno para los paquetes que tengan como destino el puerto 80 y tengan como protocolo de transporte 'nw-proto' TCP, y otra igual pero con protocolo UDP. De este modo, cualquier paquete enviado al puerto 80, sea UDP o TCP, generara que se dropee el paquete.

Para la segunda regla, establecimos un posible match en el que especificamos como capa de red IP, como protocolo de transporte 'nw-proto' UDP, el puerto de destino 'tp-dst' 5001 y como IP de origen 'nw-src' la del host 1. Creando este match, pudimos bloquear los paquetes que cumplan esta regla.

Figura 1: Topología resultante al crearla con 3 switches, usando el switch 1 como 'Firewall'



Para la tercer regla, bloqueamos cualquier comunicación IP entre los 2 hosts recibidos como parámetro, contemplamos tanto el caso en el que alguno de los hosts funcione como destinatario, como el caso en el que funcione como receptor.

4. Pruebas

En esta sección incluiremos capturas y descripción de algunas de las pruebas realizadas para verificar que las reglas pedidas en la consigna se estaban aplicando correctamente y el firewall estaba funcionando.

Para todas las pruebas se utilizó la siguiente 2 topología de red. Es importante aclarar que el SWITCH que tiene el firewall configurado es el 'S2'

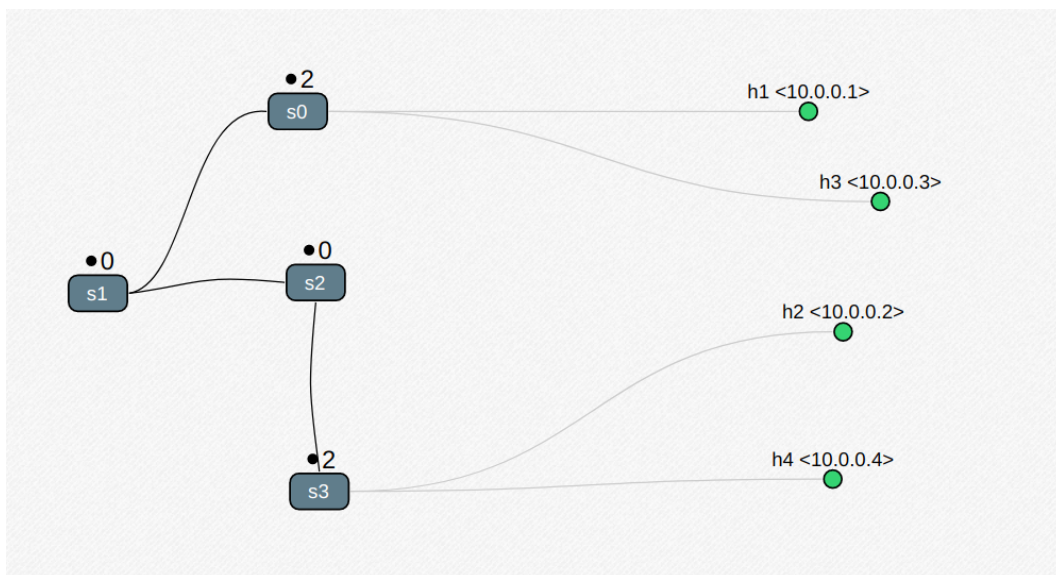


Figura 2: Imagen de la topología de red mostrando hosts y switches.

Así mismo para una mayor facilidad de interpretación del lector se dejan a continuación las

interfaces de la topología.

```
h1-eth0<->s0-eth1 (OK OK)
h3-eth0<->s0-eth2 (OK OK)
s0-eth3<->s1-eth1 (OK OK)
s1-eth2<->s2-eth1 (OK OK)
s2-eth2<->s3-eth1 (OK OK)
s3-eth2<->h2-eth0 (OK OK)
s3-eth3<->h4-eth0 (OK OK)
```

4.1. 1: Pingall Wireshark desde H1

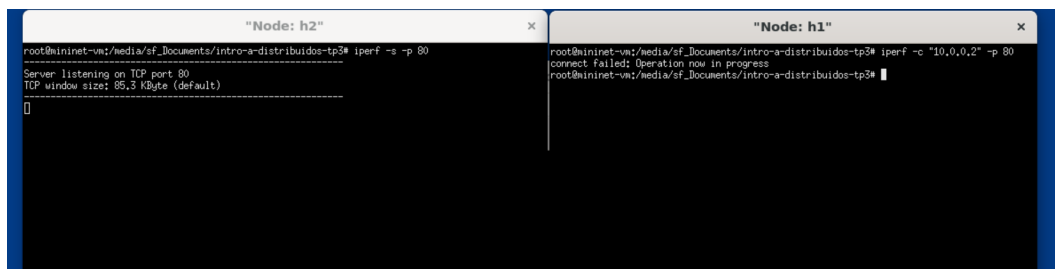
Para una primera prueba se procedió a realizar un ‘pingall’ a traves de la consola de mininet y ver cual era el flujo de paquetes desde la interfaz ‘s0-eth1’. De esta manera podíamos validar que el Host 1 estuviese interactuando correctamente con el resto de la red. Se pueden observar los paquetes ICMP saliendo y llegando desde y hacia el H1.

Capturing from s0-eth1 (on mininet-vm)									
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help									
Apply a display filter ... <Ctrl-/>									
No.	Time	Source	Destination	Protocol	Length	Info			
1	0.000000000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request	id=0x08d7, seq=1/256, ttl=64	(reply in 2)	
2	0.072065087	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply	id=0x08d7, seq=1/256, ttl=64	(request in...)	
3	0.085275902	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request	id=0x08d9, seq=1/256, ttl=64	(reply in 4)	
4	0.105754570	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) reply	id=0x08d9, seq=1/256, ttl=64	(request in...)	
5	0.114010125	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request	id=0x08db, seq=1/256, ttl=64	(reply in 6)	
6	0.128005056	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply	id=0x08db, seq=1/256, ttl=64	(request in...)	
7	0.172528881	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request	id=0x08dd, seq=1/256, ttl=64	(reply in 8)	
8	0.172585573	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply	id=0x08dd, seq=1/256, ttl=64	(request in...)	
9	0.338789835	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request	id=0x08e3, seq=1/256, ttl=64	(reply in 1..)	
10	0.338813818	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply	id=0x08e3, seq=1/256, ttl=64	(request in...)	
11	0.504312438	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request	id=0x08e9, seq=1/256, ttl=64	(reply in 1..)	
12	0.504349250	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply	id=0x08e9, seq=1/256, ttl=64	(request in...)	

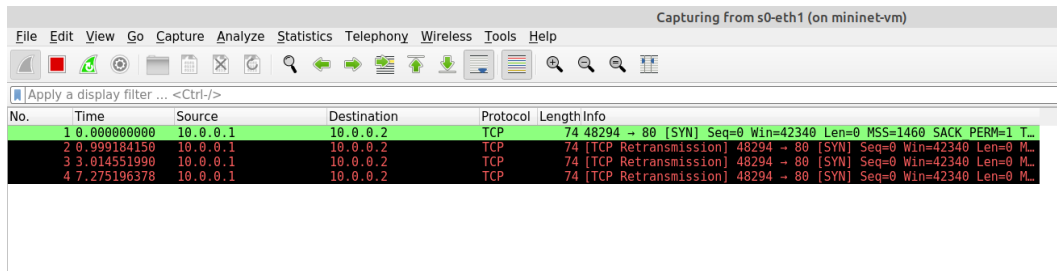
4.2. 2: Bloqueo de puerto 80 H1-H2

La segunda prueba que se documenta esta relacionada con la regla de descartar todos los mensajes cuyo destino sea el puerto 80. Para esto se levanto un servidor TCP en el H2 utilizando el comando ‘iperf’ y se procedio a realizar peticiones desde H1. Es importante visualizar que en ese flujo, S2 (quien posee el firewall) es un intermediario.

A continuación se visualiza como queda el entorno y las respuestas que se reciben.

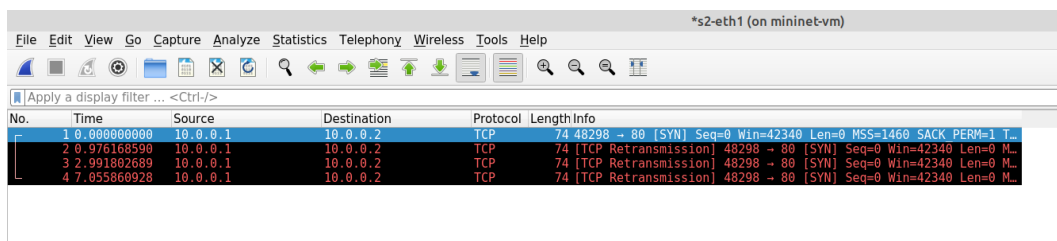


Bajo esta situación procedimos a capturar los paquetes en ‘s0-eth1’ validando efectivamente que la petición salia con éxito y las correspondientes retransmisiones de TCP.



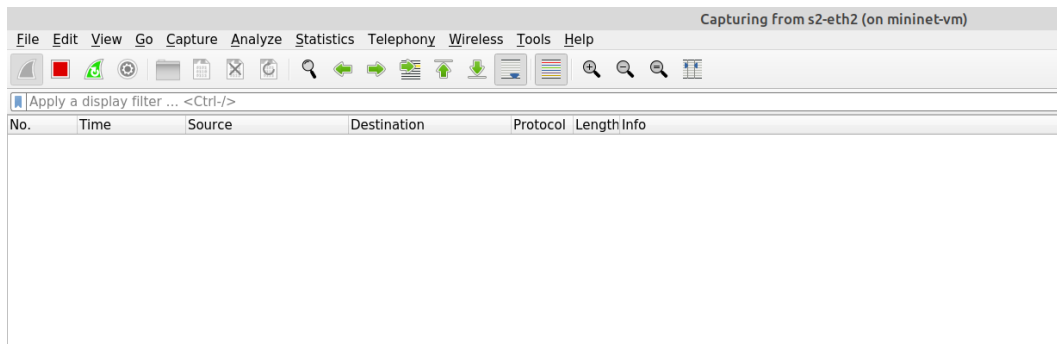
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.2	TCP	74	48294 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK PERM=1 T...
2	0.999184150	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] 48294 → 80 [SYN] Seq=0 Win=42340 Len=0 M...
3	3.014551990	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] 48294 → 80 [SYN] Seq=0 Win=42340 Len=0 M...
4	7.275196378	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] 48294 → 80 [SYN] Seq=0 Win=42340 Len=0 M...

Luego se procedió a ver si dichas peticiones llegaban a la interfaz del Firewall 's2-eth1', viendo que efectivamente sucedía.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.2	TCP	74	48298 → 80 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK PERM=1 T...
2	0.976168590	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] 48298 → 80 [SYN] Seq=0 Win=42340 Len=0 M...
3	2.991802689	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] 48298 → 80 [SYN] Seq=0 Win=42340 Len=0 M...
4	7.055860928	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] 48298 → 80 [SYN] Seq=0 Win=42340 Len=0 M...

Y para finalizar se capturo la interfaz de salida del Firewall corroborando efectivamente que los paquetes se filtraban.



No.	Time	Source	Destination	Protocol	Length	Info
-----	------	--------	-------------	----------	--------	------

4.3. 3: Bloqueo de paquetes del H1 con puerto 5001 y protocolo UDP

La siguiente prueba tiene como objeto mostrar la regla que descarta los paquetes provenientes de H1 y que utilizan el puerto 5001 bajo el protocolo UDP. Para esto lo primero que se hizo fue probar en comunicar H1 con H3. Bajo este flujo los paquetes no se dropean debido a que el Firewall no es un intermediario. Así quedó el entorno.

```

"Node: h3"
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -s -p 5001
Server listening on UDP port 5001
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)

[ 5] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 37377
[ ID] Interval      Transfer      Bandwidth      Jitter      Lost/Total Datagrams
[ 5] 0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec  0.037 ms    0/ 892 (0%)
[ 5] 0.000-3.3659 sec  2 datagrams received out-of-order

"Node: h1"
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0.0.2" -p 5001
connect failed: Connection refused
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0.0.3" -p 5001 -u
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)

[ 5] local 10.0.0.1 port 37377 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer      Bandwidth
[ 5] 0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 5] Sent 892 datagrams
[ 5] Server Report:
[ 5] 0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec  0.037 ms    0/ 892 (0%)
[ 5] 0.000-3.3659 sec  2 datagrams received out-of-order
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3#

```

Y efectivamente se verifica que en la interfaz que llega a H3 ('s3-eth3') los paquetes se capturan.

Capturing from s0-eth2 (not (tcp port 52110 and host 192.168.56.1 and tcp port 22 and host 192.168.56.103)) (on mininet-vm)

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.3	TCP	74	34504 → 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1
2	0.000030687	10.0.0.3	10.0.0.1	TCP	54	5001 → 34504 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	7.049376890	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
4	7.064231628	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
5	7.065555806	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
6	7.083745579	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
7	7.091426010	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
8	7.103318798	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
9	7.114566819	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
10	7.125811838	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
11	7.137054401	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
12	7.148648832	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
13	7.159989603	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
14	7.171112809	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
15	7.182960874	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
16	7.193329103	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
17	7.206414502	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
18	7.215575410	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
19	7.226780539	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
20	7.237964760	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
21	7.249286288	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
22	7.260472504	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470
23	7.271733270	10.0.0.1	10.0.0.3	UDP	1512	44899 → 5001 Len=1470

Luego la prueba debía verificar que entre H1 y H2 no hubiese flujo de paquetes bajo esa regla. Para esto se realizó un req del tipo TCP y uno de UDP validando que el primero llegaba y el segundo no. Así quedó el entorno.

```

"Node: h1"
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0
connect failed: Operation now in progress
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0
connect failed: Operation now in progress
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0
connect failed: Operation now in progress
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0
0.0.2" -p 5001
connect failed: Connection refused
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.
0.0.2" -p 5001 -u
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)

[ 5] local 10.0.0.1 port 39771 connected with 10.0.0.2 port 5001
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
[ ID] Interval      Transfer      Bandwidth
[ 5] 0.0-10.0 sec  1.25 MBytes  1.05 Mbits/sec
[ 5] Sent 892 datagrams
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3#

```

Se puede verificar que la interfaz conectada a H2 ('s3-eth2') recibe los paquetes TCP pero no los UDP.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.2	TCP	74	35690 → 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK PERM=1...
2	0.000039164	10.0.0.2	10.0.0.1	TCP	54	5001 → 35690 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

4.4. 4: Bloqueo de paquetes entre H1 y H4

La siguiente prueba tiene como objeto mostrar la regla que previene todo tipo de comunicación entre dos hosts. En nuestro caso se eligió H1 y H4.

Para esto se realizaron pruebas utilizando nuevamente 'iperf' en los modos cliente/servidor tanto en H1 como en H4. A continuación se muestra como queda el entorno.

```

"Node: h4"
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -s -p 3000
Server listening on TCP port 3000
TCP window size: 65.3 KByte (default)

root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -s -p 3000
Server listening on UDP port 3000
Receiving 1470 byte datagrams
UDP buffer size: 208 KByte (default)

"Node: h1"
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0.0.4" -p 3000
connect failed: Operation now in progress
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3# iperf -c "10.0.0.4" -p 3000 -u
Client connecting to 10.0.0.4, UDP port 3000
Sending 1470 byte datagrams, IPG target: 11215.21 us (kcalman adjust)
UDP buffer size: 208 KByte (default)

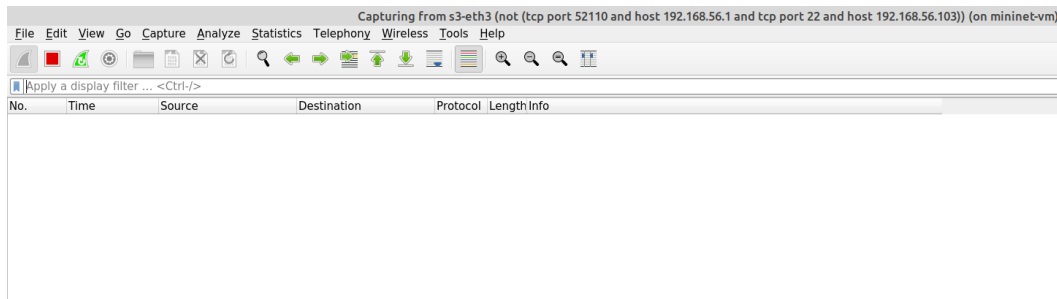
[ 5] local 10.0.0.1 port 56916 connected with 10.0.0.4 port 3000
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
[10] Interval Transfer Bandwidth
[ 5] 0.0-10.0 sec 1.25 MBytes 1.05 Mbits/sec
[ 5] Sent 892 datagrams
root@mininet-vm:/media/sf_Documents/intro-a-distribuidos-tp3#

```

En este caso primero se valido que tanto las peticiones desde H1 a H4 lleguen a la interfaz del Firewall ('s2-eth1'). Verificando que efectivamente esto sucedía.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.4	TCP	74	33244 → 3000 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK PERM=1...
2	1.014829508	10.0.0.1	10.0.0.4	TCP	74	[TCP Retransmission] 33244 → 3000 [SYN] Seq=0 Win=42340 Len=0...
3	3.030368123	10.0.0.1	10.0.0.4	TCP	74	[TCP Retransmission] 33244 → 3000 [SYN] Seq=0 Win=42340 Len=0...
4	7.286416092	10.0.0.1	10.0.0.4	TCP	74	[TCP Retransmission] 33244 → 3000 [SYN] Seq=0 Win=42340 Len=0...
5	27.245404343	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
6	27.253505926	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
7	27.258459002	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
8	27.260329801	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
9	27.261266026	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
10	27.263809232	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
11	27.264374553	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
12	27.267714844	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
13	27.278463673	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
14	27.289239325	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
15	27.302622657	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
16	27.312133486	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
17	27.322721215	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
18	27.334086135	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
19	27.345363122	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
20	27.356566794	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
21	27.367922730	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
22	27.379201815	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other
23	27.390388178	10.0.0.1	10.0.0.4	DIS	1512	PDType: 0 Other

Y luego validamos que el firewall filtraba dichos paquetes. Para esto se miro la interfaz de llegada a H4 ('s3-eth3').



5. Preguntas a responder

¿Cuál es la diferencia entre un Switch y un Router? Que tienen en comun?

La función del router es conectar distintas redes, y puede tener varios switches dentro. Mientras tanto, la función del switch es conectar distintos dispositivos. Además, el router funciona en la capa de red mientras que un switch funciona en la capa de enlace. El switch forwarda los paquetes usando MAC Addresses y el router usando direcciones de red. Otra diferencia es que los switches convencionales son 'Plug and Play', es decir, no es necesario configurarlos, mientras que los routers se tienen que configurar para que puedan funcionar.

Ambos tienen buffers para poder almacenar la información que van a procesar, el switch almacena frames mientras que el router almacena datagramas.

¿Cuál es la diferencia entre un switch convencional y un switch OpenFlow?

La diferencia está en que un switch convencional funciona independientemente del resto de la red, mientras que un switch OpenFlow al recibir un paquete que no tiene un 'flow' para el paquete, va a contactar al controlador SDN para determinar qué hacer con el paquete. El controlador podrá descargarle un flow al switch, y de ahí en adelante podrá switchear paquetes futuros similares.

El switch OpenFlow al funcionar en conjunto con un servidor(SDN) que conoce todo el layout de la red y que toma las decisiones de switching, va a poder tener nuevas capacidades. A gran escala usar OpenFlow permitirá una mayor flexibilidad, pudiendo desarrollar nuevos servicios usando OpenFlow Actions. Por ejemplo, una de las ventajas que puede tener un switch OpenFlow es que el controlador SDN puede enviar paquetes que no son críticos por caminos más largos, liberando así el tráfico en los caminos más cortos para paquetes más prioritarios.

¿Se pueden reemplazar todos los switches de la internet por switches OpenFlow? Piense en el escenario interases para elaborar su respuesta

Por un lado, reemplazar todos los switches de la internet implicaría un costo muy grande, ya que se debería reemplazar una cantidad enorme de equipos. Si estos switches usaran el mismo controlador OpenFlow, claramente se saturaría porque el controlador va a recibir una enorme cantidad de tráfico. Si el costo económico no fuera un problema y se usaran varios controladores OpenFlow, igualmente el rendimiento no sería el mismo. Esto es así porque un switch 'convencional' está diseñado de una manera simple, para poder dar una respuesta a la brevedad, mientras que usar controladores OpenFlow generaría tablas gigantes que se deberían consultar, por la enorme cantidad de switches, provocando así que el rendimiento no sea bueno. Sumado a esto, no encontramos ninguna prueba que se haya realizado en redes de gran tamaño, generando una gran incertidumbre en el caso de que se quiera realizar.

6. Dificultades encontradas

6.1. Primer encuentro con las herramientas

En un inicio fue un poco difícil familiarizarse con las herramientas de mininet, iperf y POX. La documentación de las mismas no es muy clara, y la cantidad de información que se encuentra en internet no es abundante. A partir de entender y poder probar un primer flujo sencillo, se hizo mas llevadero realizar el resto del TP.

6.2. UDP

Al desarrollar las reglas del firewall comenzamos probando con algunas básicas que bloqueen el puerto 80. Al crear el match para TCP no tuvimos problemas y pudimos corroborar con iperf que los paquetes que tenían como destino dicho puerto se dropeaban. Para filtrar el tráfico UDP duplicamos el match que teníamos y le modificamos el campo `tp_proto`. En un principio creíamos que esto no nos funcionaba ya que cuando corríamos iperf usando UDP la terminal indicaba que había paquetes que se enviaban. Creíamos que había algo en el match que estábamos creando que no era correcto.

Finalmente decidimos analizar el envío de paquetes con wireshark y logramos ver que efectivamente los paquetes UDP que iban al puerto 80 si se estaban dropeando y que el firewall estaba funcionando correctamente.

Habíamos malinterpretado la respuesta de iperf al usar UDP. La misma solo nos indicaba que los mensajes se enviaban. Esto era correcto, ya que justamente UDP no es un protocolo confiable. Se envían los paquetes y listo. El emisor no asegura que los paquetes lleguen a destino.

7. Conclusiones

Como conclusión podemos decir que pudimos notar la flexibilidad que proporcionan los controladores de OpenFlow, pudiendo controlar el flujo de manera dinámica. Si bien al principio nos costo poder utilizar el controlador POX, debido a que la documentación de la API en Python nos resulto un tanto engorrosa, una vez que pudimos concretar la primera regla nos familiarizamos con la herramienta y nos resulto mas simple hacer las reglas restantes. Ya familiarizados con la herramienta, pudimos ver la simplicidad con la cual se puede manipular el flujo de la red simulada, y la ventaja del uso de controladores OpenFlow.

Además, nos fue de utilidad poder tener un trabajo práctico en el que podamos armar una topología de red(simulada), ya que, es un tema central de la materia y nos sirvió para cerrar algunos conceptos de manera mas práctica de como se manejan las redes, haciendo un trabajo que cubra desde niveles inferiores hasta la capa de transporte.