

REDES

(TA048) CURSO 1 - ALVAREZ HAMELIN

Informe Trabajo Práctico

TP N°2: : Software-Defined Networks

24 de junio de 2025

Nicolás Grüner
110835

Bautista Boeri
110898

Victoria Fernandez
Delgado
110545

Franco Altieri Lamas
105400

Alexander
Emanuel Condo
94774

Ignacio Bertolini
104038

1. Introducción

En el contexto del avance constante de internet y la necesidad de redes más dinámicas, eficientes y adaptables, surge el concepto de Software Defined Networking (SDN). Proponiendo una separación del plano de control y el plano de datos para poder lograr sus objetivos. El protocolo OpenFlow permite a los controladores definir el comportamiento de los routers y switches.

El presente trabajo práctico tiene como objetivo principal la construcción de una topología mediante Mininet, utilizando el protocolo OpenFlow para poder implementar un Firewall a nivel de capa de enlace. Para ello, se diseñará una red controlada por el framework POX y se utilizarán herramientas como Wireshark e iperf para la verificación y análisis del correcto funcionamiento de la red y el Firewall implementado.

2. Supuestos e Hipótesis

- Las reglas de dropeo no se aplican a ipv6.
- Protocolos soportados: tcp (transporte), udp (transporte), icmp (red)
- Las reglas de droppeo deben ser consistentes. Ejemplo de mal uso: icmp (protocolo) con puerto destinado seteado.

3. Implementación

Para implementar el firewall, se creó un componente de POX. Al correr POX se puede especificar que componentes usar, y mediante la API de Python, se agregan listeners a los distintos eventos.

En el caso del firewall solo se escucha el evento 'ConnectionUp'. Esto se da cuando un switch openflow se conecta con el controlador. Al detectar este evento, se crea una instancia del firewall dedicada a esta conexión, es decir una instancia por conexión.

Ni bien se crea la instancia, se envían todas las reglas asociadas al switch de la conexión. Los switches se identifican mediante un ID (dpid). Las reglas se instalan mediante mensajes 'flow_mod' de OpenFlow. Estos mensajes envían flujos a instalar en las tablas de los switches.

El resultado es que el controlador instala las reglas proactivamente en los switches ni bien se conectan, por lo que pueden tomar decisiones sobre las mismas sin necesidad de consultar al controlador. La arquitectura de tener una instancia de firewall por conexión facilita agregar otros listeners dedicados a cada switch, por ejemplo un listener para el evento 'PacketIn'. Sin embargo, esto no resultó necesario en esta implementación.

4. Pruebas

Para corroborar el correcto funcionamiento del firewall implementado, se utilizaron las herramientas iperf y Wireshark. A través de iperf, se generó tráfico entre los distintos hosts de la topología, permitiendo comprobar si las reglas de bloqueo definidas en el archivo JSON se aplicaban correctamente. Por otro lado, Wireshark se utilizó para capturar y analizar los paquetes en la red, verificando que el tráfico no permitido efectivamente no se transmitiera, y que los paquetes permitidos circularan sin inconvenientes.

A continuación, se muestra cómo funciona el firewall ante paquetes que cumplen las 3 reglas pedidas en el enunciado.

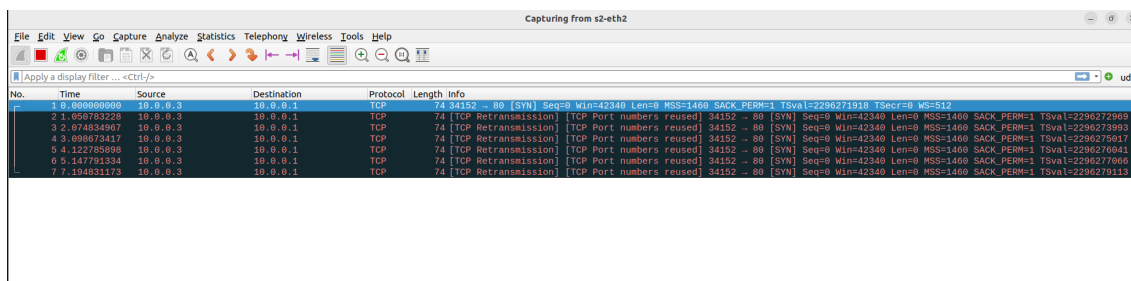
4.1. Bloqueo de tráfico con puerto destino 80 (HTTP).

```
mininet> h1 iperf -s -p 80&
mininet> h3 iperf -c h1 -p 80
tcp connect failed: Connection timed out

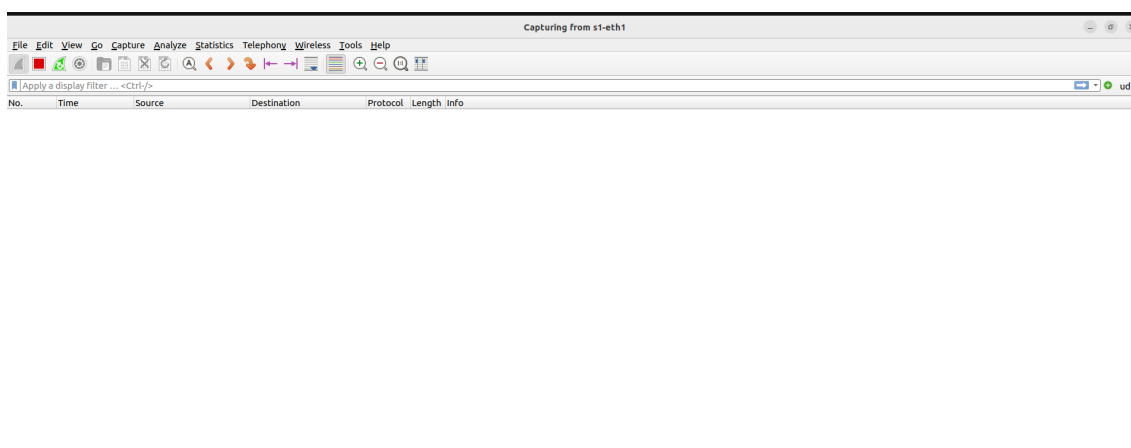
-----
Client connecting to 10.0.0.1, TCP port 80
TCP window size: -1.00 Byte (default)
-----

[ 1] local 0.0.0.0 port 0 connected with 10.0.0.1 port 80
mininet>
```

Iperf de h3 a h1 con TCP en puerto 80.



Captura de Wireshark tomada desde h3.



Captura de Wireshark tomada desde h1.

En estas imágenes se ve como se intentó establecer una conexión en el puerto 80 desde h3 hacia h1 y el resultado de esta fue “Connection timed out”, indicando que el tráfico hacia el puerto 80 fue correctamente bloqueado por el firewall.

Además, en las capturas de Wireshark se puede ver como desde h3 se envían múltiples intentos de conexión TCP que no reciben respuestas, demostrando que los paquetes están siendo descartados por el switch al aplicar la regla del firewall.

En la captura desde h1 se puede ver que no aparece ningún paquete TCP al puerto 80. Esto significa que al host nunca le llegan los intentos de conexión de h3, corroborando el correcto funcionamiento del firewall.

4.2. Bloqueo de paquetes UDP desde el host 1 hacia el puerto 5001.

4.2.1. Correcto funcionamiento con puerto 5000.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
2	0.01131125	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
3	0.011349600	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
4	0.022546982	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
5	0.033895896	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
6	0.045169677	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
7	0.056357131	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
8	0.067683135	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
9	0.078829487	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
10	0.089932335	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
11	0.101258730	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
12	0.112482132	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
13	0.123688226	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
14	0.134883539	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
15	0.146185739	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
16	0.157332467	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
17	0.168542715	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
18	0.179763141	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
19	0.190862169	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
20	0.202095975	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
21	0.213375029	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
22	0.224637178	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
23	0.235842407	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
24	0.247052940	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
25	0.258316618	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
26	0.269585155	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
27	0.280876910	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
28	0.291782853	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
29	0.303125966	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
30	0.314327556	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470
31	0.325522087	10.0.0.1	10.0.0.3	UDP	1512	40544 → 5001 Len=1470

Captura de Wireshark tomada desde h1.

```
mininet> h3 iperf -s -p 5000 -u&
mininet> h1 iperf -c h3 -p 5000 -u

-----
Client connecting to 10.0.0.3, UDP port 5000
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[  1] local 10.0.0.1 port 59280 connected with 10.0.0.3 port 5000
[ ID] Interval           Transfer         Bandwidth
[  1] 0.0000-10.0153 sec   1.25 MBytes    1.05 Mbits/sec
[  1] Sent 896 datagrams
[  1] Server Report:
[ ID] Interval           Transfer         Bandwidth          Jitter    Lost/Total Datagrams
[  1] 0.0000-10.0142 sec   1.25 MBytes    1.05 Mbits/sec    0.009 ms 0/895 (0%)
mininet>
```

Iperf de h1 a h3 con UDP en puerto 5000.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
2	0.019485076	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
3	0.021613964	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
4	0.032778607	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
5	0.043955582	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
6	0.055146500	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
7	0.066336627	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
8	0.077682019	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
9	0.088885813	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
10	0.100183992	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
11	0.111386970	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
12	0.122545606	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
13	0.133766486	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
14	0.144977632	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
15	0.156296483	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
16	0.167385116	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
17	0.178612192	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
18	0.189843786	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
19	0.201051754	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
20	0.212282938	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
21	0.223517110	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
22	0.234683644	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
23	0.245915727	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
24	0.257128940	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
25	0.268331559	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
26	0.279584632	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
27	0.290885884	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
28	0.302111958	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
29	0.313335673	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
30	0.324524715	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470
31	0.335741243	10.0.0.1	10.0.0.3	UDP	1512	59280 → 5000 Len=1470

Captura de Wireshark tomada desde h3.

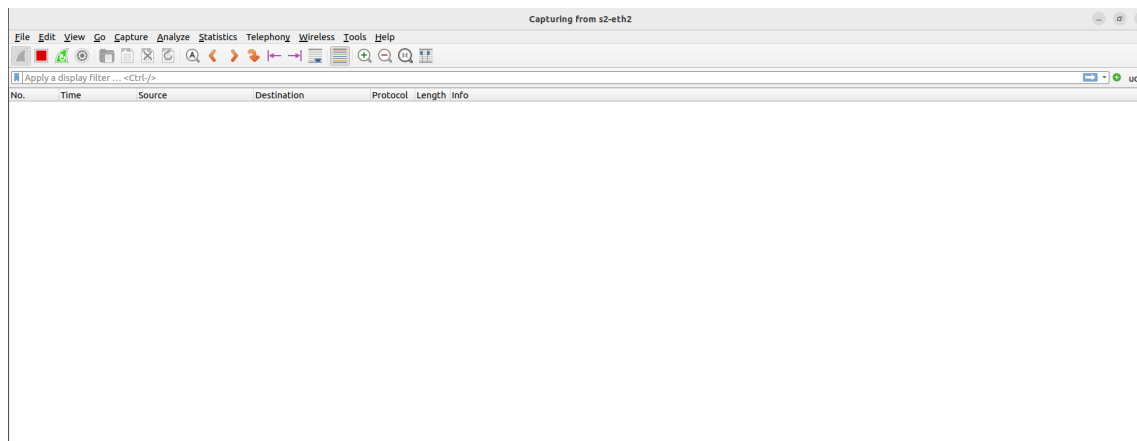
Se puede observar como al modificar el puerto, los paquetes se envían y se reciben correctamente. El funcionamiento es el esperado, ya que solo debería bloquear el tráfico del protocolo UDP de h1 que se dirija al puerto 5001. En este caso, cumple con las dos primeras, pero con la última no.

4.2.2. Funcionamiento con puerto 5001.

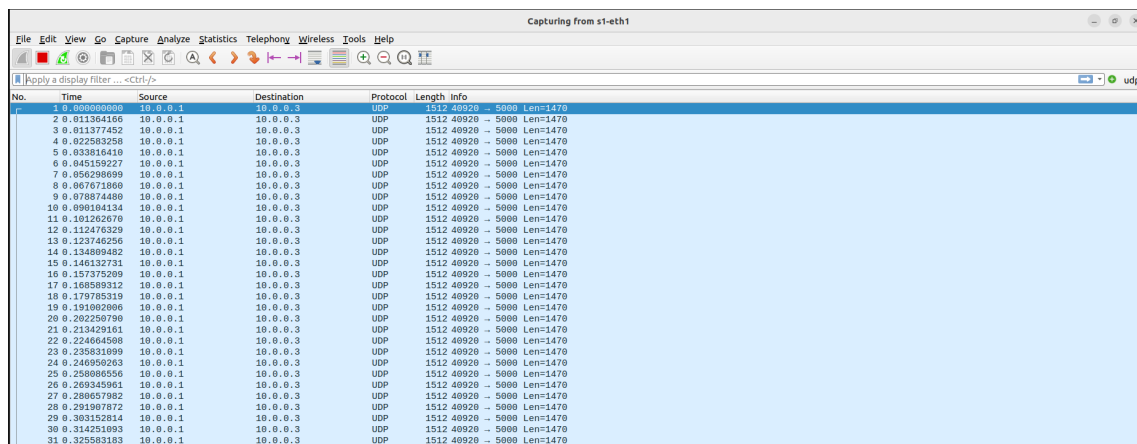
```
mininet> h3 iperf -s -p 5001 -u&
mininet> h1 iperf -c h3 -p 5001 -u

-----
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)
-----
[ 1] local 10.0.0.1 port 55916 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0153 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 3] WARNING: did not receive ack of last datagram after 10 tries.
mininet>
```

Iperf de h1 a h3 con UDP en puerto 5001.



Captura de Wireshark tomada desde h3.



Captura de Wireshark tomada desde h1.

En este caso, el puerto utilizado es el 5000. Se puede ver como efectivamente se cumple la regla mencionada anteriormente en su totalidad. Se puede notar como h1 manda los paquetes, pero el switch los bloquea gracias al firewall implementado. Es por esto, que cuando escuchamos el tráfico del lado de h3, no capturamos tráfico.

4.3. Restricción total de comunicación entre dos hosts seleccionados.

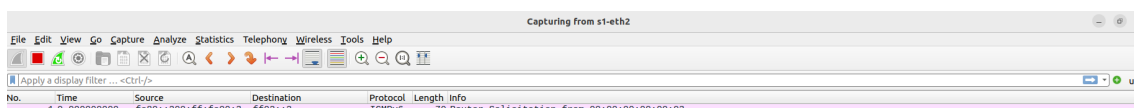
En los siguientes escenarios, se envían paquetes entre h1 y h2 utilizando UDP y TCP. Recordemos que, se aplicó una regla en el firewall para bloquear cualquier comunicación entre dos hosts elegidos. En este caso, se escogió h1 y h2, pero esto es fácilmente modificable en el JSON creado.

A continuación se presentan capturas en donde efectivamente no hay tráfico entre h1 y h2 a pesar de que los paquetes están siendo enviados.

4.3.1. Funcionamiento utilizando protocolo UDP

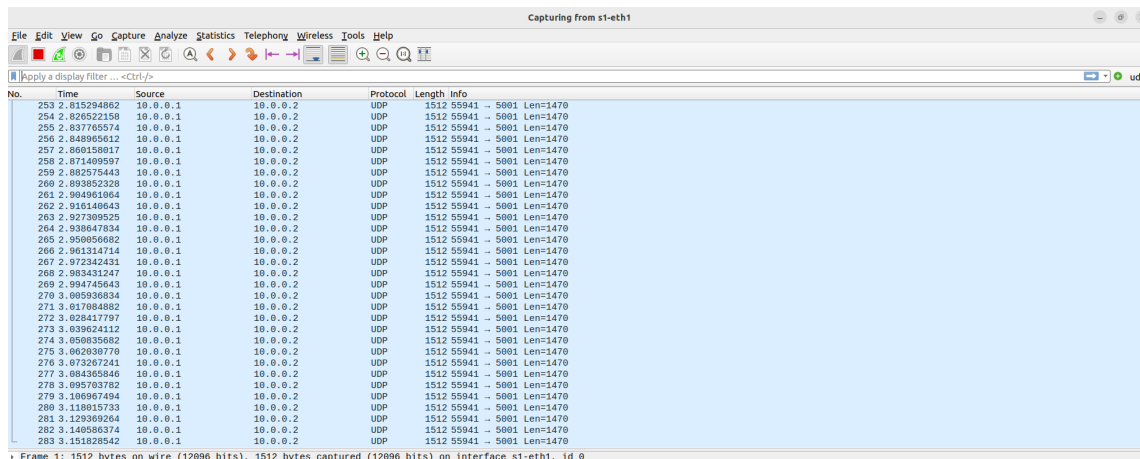
```
mininet> h2 iperf -s -u&
mininet> h1 iperf -c h2 -u&
mininet> h1 iperf -c h2 -u
-----
Client connecting to 10.0.0.2, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[  1] local 10.0.0.1 port 55941 connected with 10.0.0.2 port 5001
[ ID] Interval          Transfer      Bandwidth
[  1] 0.0000-10.0155 sec  1.25 MBytes  1.05 Mbits/sec
[  1] Sent 896 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
-----
```

Iperf de h1 a h2 con UDP en puerto 5001.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::200:ff:fe00:3	ff02::2	ICMPv6	70	Router Solicitation from 08:00:00:00:00:03

Captura de Wireshark tomada desde h2.



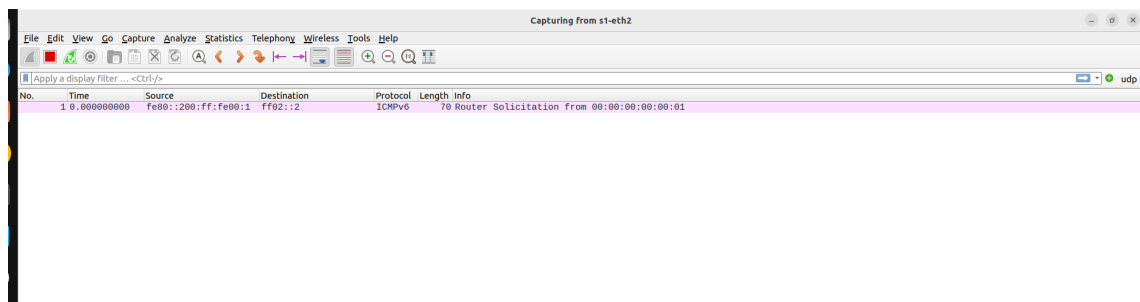
No.	Time	Source	Destination	Protocol	Length	Info
253	2.815294862	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
254	2.826522158	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
255	2.837765574	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
256	2.848995612	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
257	2.860158817	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
258	2.871409597	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
259	2.882575443	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
260	2.893852328	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
261	2.904961064	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
262	2.916140643	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
263	2.927309525	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
264	2.938647834	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
265	2.950059682	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
266	2.961314714	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
267	2.972342431	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
268	2.9834311247	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
269	2.994745643	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
270	3.005936834	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
271	3.017084882	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
272	3.028411797	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
273	3.039624112	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
274	3.050835682	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
275	3.062038770	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
276	3.073267241	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
277	3.084365846	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
278	3.095703782	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
279	3.106967494	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
280	3.118015733	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
281	3.129369264	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
282	3.140586374	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470
283	3.151828542	10.0.0.1	10.0.0.2	UDP	1512	55941 → 5001 Len=1470

Captura de Wireshark tomada desde h1.

4.3.2. Funcionamiento utilizando protocolo TCP

```
mininet> h2 iperf -s&
mininet> h1 iperf -c h2
tcp connect failed: Connection timed out
-----
Client connecting to 10.0.0.2, TCP port 5001
TCP window size: -1.00 Byte (default)
-----
[ 1] local 0.0.0.0 port 0 connected with 10.0.0.2 port 5001
```

Iperf de h1 a h2 con TCP en puerto 5001.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::200:ff:fe00:1	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:00:00:01

Captura de Wireshark tomada desde h2.

The screenshot displays a Wireshark network traffic capture titled "Capturing on st-eth1". The interface includes a menu bar, toolbar, and a packet list pane at the top. Below the packet list, the packet details pane shows the selected packet's structure, and the packet bytes pane shows the raw data.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.0.0.1	10.0.0.2	TCP	60	74 [TCP Seq=5891] Seq=5891 Len=0 MSS=1460 SACK_PERM=1 TSval=3814837424 TSecr=0 WS=512
2	0.28738608	fe80::200:faff:fe03::f	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:00:00:03
3	0.52393741	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814837424
4	0.80691114	fe80::200:faff:fe03::f	ff02::2	ICMPv6	70	Router Solicitation from 52:93:b2:7f:a0:c2
5	0.8203358	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814839350
6	0.235326481	fe80::200:faff:fe03::f	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:00:00:04
7	0.33094325	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814840052
8	0.33094325	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814840154
9	0.382349676	fe80::a0bc:gaff:fe02::b	ff02::2	ICMPv6	70	Router Solicitation from a2:9c:ba:6e:77:9f
10	0.5148971384	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814840257
11	0.519205975	fe80::200:faff:fe03::f	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:00:00:02
12	0.5605056	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814840402
13	0.7668970212	fe80::a0bc:gaff:fe02::b	ff02::fb	MNLS	221	Standard query 0x8000 PTR_pgkey-hk_tcp.local, "QM" question PTR_lpp_tcp.local, "QM" question PTR_lpps_tcp.local, "QM"
14	0.7668973239	fe80::a0bc:gaff:fe02::b	ff02::fb	MNLS	221	Standard query 0x8000 PTR_pgkey-hk_tcp.local, "QM" question PTR_lpp_tcp.local, "QM" question PTR_lpps_tcp.local, "QM"
15	0.7668973239	fe80::a0bc:gaff:fe02::b	ff02::fb	MNLS	221	Standard query 0x8000 PTR_pgkey-hk_tcp.local, "QM" question PTR_lpp_tcp.local, "QM" question PTR_lpps_tcp.local, "QM"
16	0.1313760768	fe80::200:faff:fe03::f	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:00:00:01
17	0.17194793576	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814840570
18	0.3612495873	10.0.0.1	10.0.0.2	TCP	74	[TCP Retransmission] [TCP Port numbers reused] 47844 → 5891 [SYN] Seq=0 Win=24340 Len=0 MSS=1460 SACK_PERM=1 TSval=3814840704

Captura de Wireshark tomada desde h1.

5. Preguntas a responder

1. **¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?** Un switch es un conmutador de paquetes que opera en la capa 2 (capa de enlace) y reenvía tramas basándose en la dirección MAC de destino. Es un dispositivo plug-and-play, lo que significa que no necesita configuración manual para funcionar. Para evitar bucles en la red, los switches trabajan bajo una topología de árbol de expansión (spanning tree).

Un router opera en la capa 3 del modelo OSI (capa de red) y reenvía paquetes en base a la dirección IP de destino. A diferencia del switch, no está limitado a un árbol de expansión y puede manejar topologías más complejas y redundantes, eligiendo el mejor camino entre origen y destino. Además, los routers pueden ofrecer protección tipo firewall y mejor aislamiento del tráfico entre redes.

Tanto el switch como el router son dispositivos de almacenamiento y reenvío (store-and-forward). Ambos se usan para interconectar dispositivos y reenviar datos en una red. También pueden participar en la construcción de redes eficientes, pero en diferentes niveles del modelo OSI (switch en capa 2, router en capa 3).

2. ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?

Un switch convencional, toma decisiones locales basándose en la dirección MAC destino. Solo puede reenviar según reglas simples y fijas y no tiene conocimiento global de la red.

En cambio, los switches OpenFlow, están controlados por un controlador central dentro de una arquitectura SDN. Así, el switch deja de tomar decisiones por sí mismo y simplemente sigue instrucciones del controlador. Este tiene una visión global de la red, por lo que permite tomar decisiones más eficientes. Es por esto, que se pueden aplicar políticas de reenvío más complejas y eficientes basadas en distintos campos del paquete.

3. ¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow? Piense en el escenario interASes para elaborar su respuesta

Si bien los switches OpenFlow y las redes definidas por software (SDN) ofrecen un alto grado de flexibilidad y control centralizado dentro de dominios administrados por una única entidad, su adopción a gran escala en el contexto de Internet global presenta limitaciones significativas, particularmente en el plano inter-AS, donde reemplazar los routers tradicionales por switches OpenFlow no resulta viable. Internet está estructurada como una red de redes, donde cada sistema autónomo gestiona su propia infraestructura y políticas de ruteo. La comunicación entre estos dominios se establece mediante protocolos como BGP (Border Gateway Protocol), los cuales permiten a cada AS anunciar y seleccionar rutas de manera independiente, en función de acuerdos comerciales, criterios técnicos o estrategias de redundancia.

En este escenario, un enfoque centralizado como el que propone OpenFlow entraría en conflicto con la naturaleza distribuida y descentralizada del ruteo inter-AS. Implementar un controlador SDN que gestione de forma global las rutas entre todos los sistemas autónomos implicaría no solo enormes desafíos técnicos en términos de escalabilidad y confiabilidad, sino también barreras políticas y administrativas difíciles de sortear. Por lo tanto, aunque los

switches OpenFlow pueden reemplazar exitosamente a los routers tradicionales en entornos controlados (como centros de datos, redes de campus o redes corporativas), su aplicación como sustituto de los routers en el backbone de Internet, particularmente en el intercambio entre ASes, no es factible en la práctica.

6. Dificultades encontradas

Durante el desarrollo del trabajo práctico, nos encontramos con diversas dificultades:

- Utilizar POX por primera vez y entender que utilizar para aplicar las reglas del firewall fue el primer obstáculo que se nos presentó.
- Inicialmente no estábamos seguros si las reglas se debían aplicar al recibir un paquete o ya configurarlas para cada switch correspondiente al inicializar el firewall.
- Frecuentaba un mensaje de error el cuál no sabíamos de dónde podía provenir: *ERROR:packet:(dns) parsing questions: ord() expected string of length 1, but int found.*
- Debuggear resultó tedioso, ya que era necesario reiniciar el firewall, iniciar mininet, capturar el tráfico con wireshark y correr los comandos pertinentes en cada host.

7. Conclusión

A lo largo de este trabajo práctico se logró cumplir con el objetivo principal de implementar un firewall utilizando la herramienta POX, mininet y el protocolo OpenFlow. Se construyó una topología en la cual se aplicaron las distintas reglas de filtrado definidas en un archivo JSON. Esto permitió bloquear determinados tipos de tráfico según criterios como dirección IP, puerto y protocolo de transporte.

El funcionamiento del firewall fue verificado utilizando herramientas como iperf y Wireshark. Esto permitió observar como las reglas de bloqueo se aplicaban correctamente, filtrando aquellos paquetes que cumplían con las reglas definidas en el JSON.

Se logró afianzar conceptos sobre SDN, la separación del plano de datos del plano de control y del protocolo OpenFlow. Se comprendió la flexibilidad que ofrece la arquitectura al permitir modificar el funcionamiento de la red sin necesidad de intervenir en el hardware. También se comprendieron las limitaciones de las tecnologías, comparadas con la infraestructura tradicional del internet.