

REDES
(TA048) CURSO 02 - ALVAREZ HAMELIN

Trabajo Práctico 2 Software-Defined Networks

Martín González Prieto
105738

Gian Luca Spagnolo
108072

Tomás Caporaletti
108598

Helen Elizabeth Chen
110195

Martin Osan
109179

Índice

1. Introducción	3
2. Herramientas utilizadas	3
3. Implementación	3
3.1. Topología	4
3.2. Firewall	5
4. Pruebas	6
4.1. iPerf	6
4.2. WireShark	8
5. Preguntas a responder	10
5.1. ¿Cuál es la diferencia entre un Switch convencional y un Router? ¿Qué tienen en común?	10
5.2. ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?	10
5.3. ¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow? Piense en el escenario interASes para elaborar su respuesta	11
6. Dificultades encontradas	11
7. Conclusión	12

1. Introducción

El objetivo principal del trabajo es permitirnos interiorizar los conceptos particulares de Software-Defined Networks (SDN) y el protocolo OpenFlow, mediante la construcción de una **topología de red dinámica** y la implementación de un Firewall en la capa de enlace. Para ello, hemos empleado herramientas como Mininet, para simular la topología de red, y el controlador POX, que actúa como el cerebro encargado de gestionar las políticas y reglas de ruteo.

2. Herramientas utilizadas

A continuación, mencionaremos a detalle cada una de las herramientas empleadas para cumplir con la implementación previamente descrita, junto a sus respectivos comandos para poder ejecutarlas efectivamente:

- **Mininet:** Simulador de redes que permite crear y gestionar topologías de switches OpenFlow y hosts.

```
1      sudo mn --custom topology.py --topo mytopo,ammount_switches=5 --mac --  
2      arp --switch ovsk --controller remote
```

- **POX:** Un controlador SDN con una API en Python que facilita la implementación de políticas personalizadas de reenvío y seguridad.

```
1      python pox/pox.py forwarding.l2_learning firewall  
2
```

- **Wireshark:** Herramienta utilizada para analizar, registrar y comparar el tráfico de red.

- **iPerf:** Es una herramienta utilizada para pruebas de rendimiento en la red.

```
1      // Server  
2      [host] iperf -s -p [port] &  
3  
4      //Client  
5      [src_host] iperf -c [dst_host] -p [port]  
6
```

3. Implementación

La implementación se enfoca en los tres aspectos mencionados previamente: la configuración y validación de una topología de switches OpenFlow, el diseño de un Firewall y el análisis del comportamiento de la red mediante herramientas de monitoreo, las cuales se utilizan con el objetivo de validar las conexiones e identificar el tráfico correspondiente mediante las políticas implementadas.

3.1. Topología

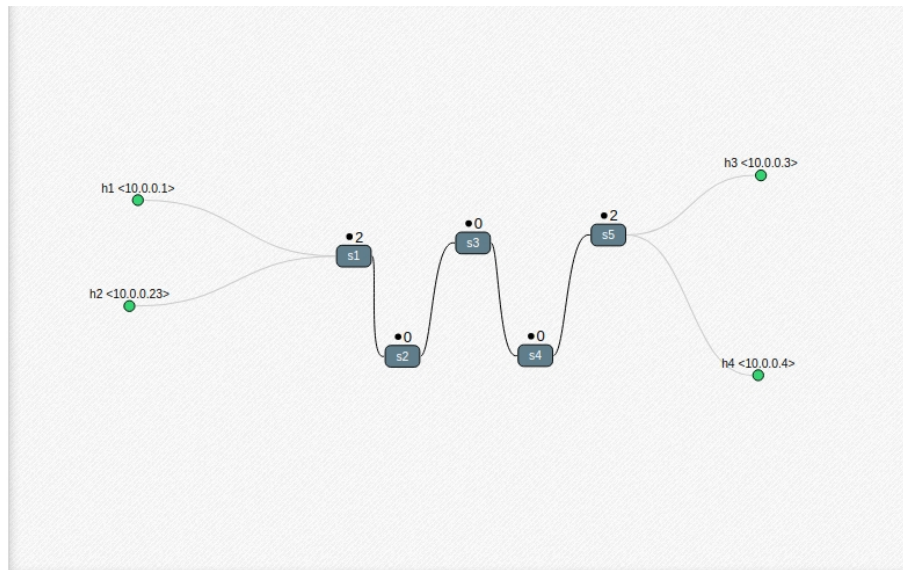


Figura 1: Visualización de la Topología Implementada

El gráfico de la Figura 1 se encuentra en este link de Spear Narmox. Nuestra topología fue implementada en Python, y recibe por parámetros una cantidad de Switches (que debe ser mayor a 1) y direcciones IP. De esta forma, busca agregar los hosts correspondientes en base a las direcciones, y los Switches respectivos enlazándolos entre si y con los hosts correspondientes, quedando implementado una topología tal cual es descrita en la Figura 1.

Como también nuestra implementación de la topología es dinámica, se puede ver en la siguiente figura una configuración con 7 switches y 4 hosts.

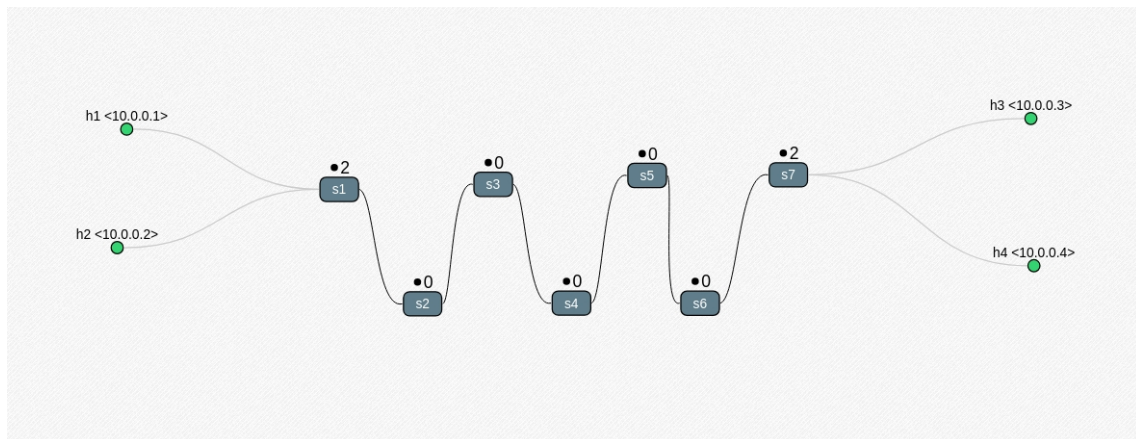


Figura 2: Visualización de la Topología Implementada con otra cantidad de switches

3.2. Firewall

En lo que respecta a la implementación del Firewall, es importante destacar que contamos con un archivo de configuración que permite agregar reglas al mismo de manera dinámica. Dichas reglas se instalarán en el switch de id=1, suponiendo que la red tendrá como mínimo un switch.

Los campos permitidos para dicha configuración son:

- **dst port** (número): Bloquea el tráfico proveniente de dicho puerto.
- **src ip** (string): Bloquea el tráfico proveniente de dicho host identificado por su IP.
- **dst ip** (string): Bloquea el tráfico que se dirige a dicho host identificado por su IP.
- **transport protocol** (string): Bloquea cualquier tráfico proveniente en dicho protocolo de transporte (UDP or TCP).
- **ip version** (string): Version de IP para identificar a los hosts (por default es IPv4)

En caso de que no se cumplan con estos parámetros, aquellos campos no se tendrán en cuenta. La configuración utilizada en este trabajo para cumplir con las reglas propuestas es la siguiente:

```
1  {
2    "rules": [
3      {
4        "dst_port": 80,
5        "ip_version": "v4",
6        "transport_protocol": "UDP"
7      },
8      {
9        "src_ip": "10.0.0.1",
10       "dst_port": 5001,
11       "transport_protocol": "UDP",
12       "ip_version": "v4"
13     },
14     {
15       "src_ip": "10.0.0.2",
16       "dst_ip": "10.0.0.4",
17       "ip_version": "v4"
18     },
19     {
20       "src_ip": "10.0.0.4",
21       "dst_ip": "10.0.0.2",
22       "ip_version": "v4"
23     }
24   ]
25 }
```

En relación al funcionamiento interno del Firewall, cabe destacar cómo se cargan dichas reglas en el controlador, para lo cual dos funciones auxiliares que sirven para esto.

Dentro de **handleConnectionUp** de la clase Firewall, la cual será invocada en el **launch()**, se realiza la carga de reglas con **loadRules(event)**. En dicha función se recorre la lista de reglas, las cuales están implementadas como diccionarios cuyos campos representan que parte de la regla contará para que se bloquee el tráfico efectivamente. En este recorrido, se va cargando dentro de una regla genérica (**genericRule**), que por default tiene sus campos en **None** exceptuando la versión de IP (por defecto, IPv4). Para esto, la función **readField(rule, field)** se encarga de que dicho diccionario contenga las instrucciones deseadas para cada argumento, o devuelva un **None** en caso de que no pueda reconocer el campo o su valor. Una vez inicializada la regla, se envía dentro de la función **event.connection.send()** para que pueda ser configurada en el switch.

```
1 def loadRules(self, event):
2     for rule in self.rules:
3         rule = genericRule(port=read_field(rule, PORT_FIELD),
4                             src_ip=read_field(rule, SRC_IP_FIELD),
5                             dst_ip=read_field(rule, DST_IP_FIELD),
6                             transport_protocol=read_field(rule,
7                                                         TRANSPORT_PROTOCOL_FIELD),
8                             ip_protocol=read_field(rule, IP_FIELD))
9         event.connection.send(rule)
10
11     log.info("Firewall rules installed on %s switch", dpidToStr(event.dpid))
12
13 def genericRule (port=None, src_ip=None, dst_ip=None, transport_protocol=None,
14                  ip_protocol=IPv4_CONFIG):
15     rule = of.ofp_flow_mod()
16     rule.match.dl_type = ip_protocol
17     rule.match.tp_dst = port
18     rule.match.nw_src = src_ip
19     rule.match.nw_dst = dst_ip
20     rule.match.nw_proto = transport_protocol
21     return rule
```

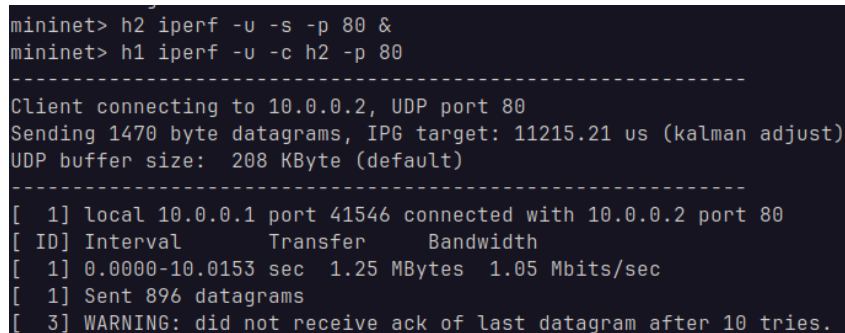
4. Pruebas

A continuación, se presentan visualizaciones correspondientes de iPerf y Wireshark para demostrar el correcto funcionamiento del FireWall implementado.

4.1. iPerf

1. Se descartan mensajes cuyo puerto destino sea 80. Comandos ejecutados en Mininet:

```
1 h2 iperf -u -s -p 80 &
2 h1 iperf -u -c h2 -p 80
3
```



```
mininet> h2 iperf -u -s -p 80 &
mininet> h1 iperf -u -c h2 -p 80
-----
Client connecting to 10.0.0.2, UDP port 80
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[  1] local 10.0.0.1 port 41546 connected with 10.0.0.2 port 80
[ ID] Interval      Transfer    Bandwidth
[  1] 0.0000-0.0153 sec 1.25 MBytes 1.05 Mbits/sec
[  1] Sent 896 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
```

Figura 3: Al ser enviado al puerto 80, la información no llega

2. Se descartan todos aquellos mensajes que provengan del host 1 que tengan como puerto destino 5001, y estén utilizando el protocolo UDP (User Datagram Protocol). Comandos ejecutados en Mininet:

```
1 h4 iperf -u -s -p 5001 &
2 h1 iperf -u -c h4 -p 5001
3
```

```
mininet> h4 iperf -u -s -p 5001 &
mininet> h1 iperf -u -c h4 -p 5001
-----
Client connecting to 10.0.0.4, 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 50993 connected with 10.0.0.4 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.
```

Figura 4: Al ser enviado desde h1 al puerto 5001, la información no llega

3. Bloqueo de comunicación entre 2 hosts cualquiera (se eligieron h2 y h4). Comandos ejecutados en Mininet:

```
1 h2 iperf -u -c h4 -p 1000
2 h4 iperf -u -c h2 -p 1000
3
```

```
mininet> h2 iperf -u -c h4 -p 1000
-----
Client connecting to 10.0.0.4, UDP port 1000
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[  1] local 10.0.0.2 port 36853 connected with 10.0.0.4 port 1000
[ 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> h4 iperf -u -c h2 -p 1000
-----
Client connecting to 10.0.0.2, UDP port 1000
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[  1] local 10.0.0.4 port 34206 connected with 10.0.0.2 port 1000
[ 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.
```

Figura 5: Se bloquean h2 y h4 entre ellos

4.2. WireShark

A continuación se muestran otras visualizaciones correspondientes de Wireshark mostrando el funcionamiento del Firewall. Los comandos usados para los casos donde el Firewall es activado son los mismos que se usaron para la visualización por iPerf

1. Primero se muestra un ejemplo del correcto funcionamiento de la topología cuando el Firewall no realiza acciones bloqueantes. Se puede ver que hay un ida y vuelta de paquetes entre h1 y h3.

No.	Time	Source	Destination	Protocol	Length	Info
22905	239.658091712	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22906	239.669332469	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22907	239.680556425	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22908	239.691753941	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22909	239.701535021	10.0.0.3	10.0.0.1	FIND	170	1001 → 38363 Len=128, Unknown (0x0000)
22910	239.712655542	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22911	239.712772963	10.0.0.3	10.0.0.1	FIND	170	1001 → 38363 Len=128, Unknown (0xffff)
22912	239.724105952	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22913	239.557127298	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22914	239.568332649	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22915	239.579564810	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22916	239.590764530	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22917	239.601978467	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22918	239.613203705	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22919	239.624418353	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22920	239.635624134	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22921	239.646839994	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22922	239.658097683	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22923	239.669337309	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22924	239.680556226	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0x0000)
22925	239.691757427	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22926	239.695856102	10.0.0.3	10.0.0.1	FIND	170	1001 → 38363 Len=128, Unknown (0x0000)
22927	239.712659519	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22928	239.712692581	10.0.0.3	10.0.0.1	FIND	170	1001 → 38363 Len=128, Unknown (0xffff)
22929	239.724105952	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22930	239.691738482	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22931	239.712636656	10.0.0.1	10.0.0.3	FIND	1512	38363 → 1001 Len=1470, Unknown (0xffff)
22932	239.713189787	10.0.0.1	10.0.0.1	FIND	170	1001 → 38363 Len=128, Unknown (0x0000)
22933	239.714889556	10.0.0.3	10.0.0.1	FIND	170	1001 → 38363 Len=128, Unknown (0xffff)

Figura 6: Intercambio correcto entre dos hosts.

2. Se descartan mensajes cuyo puerto destino sea 80. Nótese que no hay respuestas.

No.	Time	Source	Destination	Protocol	Length	Info
1384	70.172914600	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1385	70.182996214	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1386	70.193077778	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1387	70.203157989	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1388	70.213239192	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1389	70.223319163	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1390	70.233401127	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1391	70.243482340	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1392	70.253564976	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1393	70.263645488	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1394	70.273734997	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1395	70.283816881	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1396	70.293898344	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1397	70.303978696	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1398	70.314059779	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1399	70.324144939	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1400	70.334226563	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1401	70.344324798	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1402	70.354414387	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1403	70.364497153	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1404	70.374580360	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1405	70.384660823	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1406	70.394743748	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1407	70.404827403	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1408	70.414908971	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1409	70.424990880	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1410	70.435089000	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1411	70.445173132	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1412	70.455254721	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1413	70.465336460	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1414	70.475419091	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1415	70.485499207	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1416	70.495595703	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1417	70.505680498	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1418	70.515750495	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1419	70.525846891	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1420	70.535930223	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1421	70.546011781	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1422	70.556094682	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1423	70.566176481	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1424	70.576260544	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1425	70.586343435	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1426	70.596457295	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1427	70.606547149	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1428	70.616630361	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1429	70.626734873	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1430	70.636823154	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470
1431	70.646913720	10.0.0.1	10.0.0.2	UDP	1512	49464 → 80 Len=1470

Figura 7: Mensajes con puerto destino 80

3. Se descartan todos aquellos mensajes que provengan del host 1 que tengan como puerto destino 5001, y estén utilizando el protocolo UDP (User Datagram Protocol).

No.	Time	Source	Destination	Protocol	Length	Info
2647	142.892437364	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2648	142.902538222	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2649	142.912635764	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2650	142.922748894	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2651	142.932848720	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2652	142.942948817	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2653	142.953048983	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2654	142.963148538	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2655	142.973248173	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2656	142.983347418	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2657	142.993447153	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2658	143.003545066	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2659	143.013643859	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2660	143.023743655	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2661	143.033840716	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2662	143.043940010	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2663	143.054039165	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2664	143.064139852	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2665	143.074235851	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2666	143.084333042	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2667	143.094432247	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2668	143.104529687	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2669	143.114628291	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2670	143.124745680	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2671	143.134844694	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2672	143.144943537	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2673	143.155042993	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2674	143.165141867	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2675	143.175241342	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2676	143.185340817	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2677	143.195440823	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2678	143.205539907	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2679	143.215638501	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2680	143.225752874	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2681	143.235853742	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2682	143.245955992	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2683	143.256056489	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2684	143.266154250	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2685	143.276352833	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2686	143.286451166	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2687	143.296549649	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2688	143.306647641	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2689	143.316745423	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2690	143.326845309	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2691	143.336947369	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2692	143.347047075	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2693	143.357145858	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470
2694	143.367244472	10.0.0.1	10.0.0.4	UDP	1512	58916 → 5001 Len=1470

Figura 8: Mensajes que envía h1 al puerto 5001 usando UDP

4. Finalmente, se puede ver que cuando h2 trata de mandar paquetes a h4 este ultimo no devuelve nada, lo que indica que una vez más el Firewall está activo.

No.	Time	Source	Destination	Protocol	Length	Info
24009	414.004423021	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24010	414.014523665	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24011	414.024623396	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24012	414.034732385	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24013	414.044838408	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24014	414.054937477	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24015	414.065037710	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24016	414.075138563	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24017	414.085237643	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24018	414.095338347	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24019	414.105442126	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24020	414.115544051	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24021	414.125643141	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24022	414.135747961	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24023	414.145827335	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24024	414.156024922	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24025	414.166124312	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24026	414.176222100	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24027	414.186319466	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24028	414.196416984	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24029	414.206514441	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24030	414.216612579	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24031	414.226713192	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24032	414.236828513	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24033	414.246927543	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24034	414.257026703	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24035	414.267125743	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24036	414.277224793	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24037	414.287324434	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24038	414.297420990	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24039	414.307518797	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470
24040	414.317617276	10.0.0.2	10.0.0.4	UDP	1512	47941 → 1000 Len=1470

Figura 9: Ráfaga de mensajes enviada desde h2 a h4, mostrando una relación donde hay bloqueo

5. Preguntas a responder

5.1. ¿Cuál es la diferencia entre un Switch convencional y un Router? ¿Qué tienen en común?

Tanto los Switches como los Routers:

1. Conectan dispositivos importantes para la comunicación en redes.
2. Toman decisiones sobre cómo los datos deben manejarse cuando se reciben.
3. Incluyen funcionalidades de seguridad.
4. Pueden ser gestionados centralmente a través de un controlador en una arquitectura de Redes Definidas por Software (SDN).

Sin embargo, presentan diferencias en los siguientes campos:

■ **La función principal de cada uno:**

Los Switches permiten la comunicación entre diferentes dispositivos en una LAN (ya que operan en la capa 2 de enlace del modelo OSI) manejando direcciones MAC, mientras que los Routers se encargan de la comunicación entre diferentes redes (los cuales pertenecen a la capa 3 de red) usando direcciones IP.

■ **La manera de enviar los paquetes:**

Los Switches envían directamente los paquetes al dispositivo destino en la LAN, ya que cada puerto de un Switch recuerda la dirección MAC que está conectada, enviando los datos sólo al puerto correspondiente. Los Routers, por otro lado, utilizan tablas de enrutamiento y ciertos protocolos (como IP) para tomar decisiones sobre el envío de paquetes.

■ **La capacidad de Redireccionamiento y Seguridad:**

Los Switches ofrecen una seguridad básica y no redireccionan datos fuera de su red local, pero los Routers sí pueden hacerlo a través de protocolos de enrutamiento para optimizar el flujo de datos, y además pueden aplicar reglas de seguridad (como Firewalls) para bloquear o permitir tráfico entre redes, aumentando así su seguridad.

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

Switch convencional

1. Conecta varios dispositivos en una red local (LAN) y decide cómo reenviar los datos basándose en las direcciones MAC.
2. Las decisiones de reenvío son automáticas y autónomas, basadas en tablas construidas dinámicamente por el Switch mismo.
3. Es 'cerrado', lo que significa que su comportamiento está definido por el fabricante y no puede ser modificado fácilmente.

Switch OpenFlow

1. Conecta dispositivos en una red sin tomar decisiones por sí mismo. Además, tiene un alcance más amplio.
2. Sigue reglas enviadas desde un controlador SDN central (el 'cerebro' de la red) mediante el protocolo OpenFlow.
3. Es programable y permite configurar cómo manejar datos según criterios personalizados, como la dirección de origen, tipo de tráfico, u otras etiquetas.
4. Ofrece más flexibilidad porque separa el plano de control (donde se toman las decisiones) del plano de datos (donde se ejecutan las decisiones).

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

No es viable reemplazar todos los routers de Internet por switches OpenFlow.

Los routers cumplen funciones críticas que OpenFlow no aborda completamente, las cuales engloban algunas como:

1. El tráfico de los routers entre diferentes redes autónomas (AS) usando protocolos especializados como BGP (Border Gateway Protocol).
2. **El ruteo inter-AS tiene requisitos específicos:** En el escenario **inter-AS**, los routers toman decisiones basadas no solo en eficiencia (por ejemplo, eligiendo el camino más corto) sino también en políticas (como evitar ciertos países o redes en específico). Esto implica una lógica altamente compleja que no está directamente soportada por OpenFlow, el cual está más orientado a redes internas o intra-AS.
3. **Autonomía de los dispositivos:** En Internet, los routers deben ser autónomos porque no siempre es práctico depender de un controlador central. Si el controlador SDN de un switch OpenFlow falla, toda la red puede quedar inoperativa en consecuencia. Los routers tradicionales poseen suficiente inteligencia como para operar independientemente, incluso en situaciones donde hay fallos de red.
4. **Escalabilidad y complejidad:** La topología de Internet es extremadamente masiva y dinámica, lo que presenta desafíos significativos para su gestión. Aunque SDN, y en particular OpenFlow, resulta altamente útil en entornos controlados como redes dentro de un único dominio (intra-AS), sus beneficios son más difíciles de aplicar al ruteo entre múltiples dominios (inter-AS). OpenFlow está diseñado para operar con una vista completa y centralizada de la red que controla, lo cual es práctico en redes internas. Sin embargo, en el contexto global de Internet, esto implicaría manejar y estar al tanto de millones de rutas y dispositivos distribuidos a lo largo de numerosos dominios autónomos con políticas independientes, lo que dificulta su escalabilidad y viabilidad práctica.

6. Dificultades encontradas

Debido a las diferentes versiones de POX y la necesidad de utilizar una versión anterior de Python (en este caso la 2.7.18), consideramos que uno de los desafíos técnicos del trabajo fue la correcta utilización de un entorno adecuado para poder ejecutar los programas. Hemos aplicado diferentes formas, como por ejemplo haciendo directamente un downgrade de la versión de Python, pero sin lugar a dudas, la mas practica fue la utilización de **Conda** (en este caso en su versión MiniConda). Gracias a esto, fuimos capaces de activar y desactivar a gusto un entorno virtual donde se utiliza Python en la versión adecuada para ejecutar POX.

Por otro lado, la implementación de la topología correspondiente como las pruebas necesarias realizadas en las herramientas correspondientes implicaron un aprendizaje y entendimiento de cada una para poder utilizarlas sin ningún inconveniente, como se ha mostrado a lo largo de este informe.

7. Conclusión

En este trabajo práctico logramos desarrollar una topología funcional y una implementación de Firewall operativo utilizando las herramientas de Software Definidas por Red (SDN). Las pruebas realizadas demostraron que las funcionalidades implementadas cumplen con los objetivos del proyecto, y los resultados fueron validados correctamente mediante el análisis de tráfico y los registros generados. Este enfoque no solo demuestra la viabilidad de implementar redes SDN en entornos controlados, sino que también proporciona una base sólida para exploraciones futuras en el campo de la seguridad y el rendimiento de redes.