Redes

(TA048) Curso 02 Alvarez Hamelin

Trabajo Práctico
TP N°2: Software-Defined Networks

28 de Noviembre de 2024

Mateo Alvarez
108666

Martín Juan Cwikla
107923

Máximo Gismondi
110119

Juan Manuel Pascual Osorio
105916

Maximiliano Nicolas Otero Silvera
108634

Índice

1. Introducción

Este trabajo tiene como objetivo la implementación de una red de tipo SDN (Software Defined Network) que utilice el protocolo OpenFlow.

El objetivo del mismo es el diseño, simulación y experimentación de una red parametrizable, que permita modificar la topología, y la implementación de un firewall configurable, capaz de aplicar políticas específicas de bloqueo de tráfico en la capa de enlace. Asimismo, se empleará el simulador Mininet para emular el comportamiento de la red y el controlador POX como herramienta para implementar las políticas de control necesarias.

Mediante este trabajo, se busca profundizar en los conceptos de las SDN, explorando su flexibilidad y su capacidad para introducir políticas específicas en el manejo del tráfico. Además, se pretende evaluar la performance y funcionalidad del entorno construido, integrando herramientas como Wireshark e iPerf para la verificación y el análisis de los resultados obtenidos.

2. Implementación

En este trabajo se diseñaron e implementaron algunos componentes para lograr una simulación de una red SDN con control de tráfico. Para ello, se utilizó el simulador Mininet y el controlador POX. A su vez, se desarrollaron implementaciones en la versión 3 de Python de una Topología configurable por parámetros y un Firewall que se configura mediante un archivo JSON.

2.1. Mininet

Mininet es una herramienta de simulación de redes ampliamente utilizada en entornos académicos y de investigación para modelar y probar topologías de red de manera eficiente. Permite crear redes virtuales completas en una sola máquina, incluyendo hosts, switches, routers y enlaces, utilizando técnicas de virtualización liviana basadas en namespaces de red en Linux.

La idea de Mininet es crear entornos de red virtuales que se comportan de manera similar a redes físicas reales. Mininet provee la virtualización de dos componentes claves: los Hosts y los Switches. Cada componente se instancia como un proceso separado en el sistema operativo y a su vez, estos procesos están conectados a través de interfaces virtuales que simulan los enlaces físicos de una red real.

Los Hosts representan las máquinas finales en la red, emuladas como procesos aislados. Pueden ejecutar aplicaciones de red reales, como servidores web o herramientas de diagnóstico (e.g., ping, iperf). Por otro lado, los switches y routers simulan dispositivos de red SDN dedicados al correcto envío y recepción de paquetes a través de la red. Más adelante, en la sección de Preguntas, se describe a fondo lo que hace cada uno y cuales son sus diferencias. En este trabajo, se utilizan switches configurados para operar con el protocolo OpenFlow.

En este trabajo, Mininet se utilizó para diseñar e implementar la topología de red. El archivo "topology.py" desarrollado, genera una red parametrizable que se adapta a las especificaciones requeridas. La red fue configurada para interactuar con el controlador POX, permitiendo implementar funcionalidades avanzadas, como la gestión de políticas de firewall para descartar paquetes específicos.

2.2. Topología Desarrollada

La topología implementada para este trabajo consiste de una red configurable, que utiliza el módulo mininet.topo. y está desarrollada en python3. La topología permite variar la cantidad de switches mediante un parámetro, adaptándose a diferentes escenarios de prueba. Los extremos de la red cuentan con dos hosts conectados a los switches inicial y final, y se establecen enlaces adicionales para simular la conectividad de la red.

Para la implementación, se desarrolló una clase llamada MyTopo que hereda de Topo, donde se configura dinámicamente el número de switches y hosts. El código se estructura de la siguiente manera:

Se instancian cuatro hosts y una cantidad variable de switches. Los hosts se identifican como h1, h2, h3 y h4, mientras que los switches se nombran de forma incremental (s1, s2, ...). Los switches operan sin dependencias adicionales entre ellos, lo que asegura mayor control sobre su comportamiento durante las simulaciones.

Luego, se crean enlaces directos entre:

1. Los hosts extremos (h1 y h2) y el primer switch (s1).
2. Los hosts finales (h3 y h4) y el último switch de la cadena.
3. Los switches intermedios, conectándolos secuencialmente para formar una cadena lineal.

La cantidad de switches puede definirse mediante el parámetro "switches" al instanciar la

topología. Si no se proporciona un valor o si se establece un número menor o igual a cero, se utiliza un valor mínimo de 1 switch.

2.3. Controlador POX

El controlador POX es un framework escrito en Python utilizado para desarrollar aplicaciones en redes SDN mediante el protocolo OpenFlow. Su propósito principal es permitir la implementación y gestión de políticas de control en switches y routers compatibles con OpenFlow.

En este trabajo, el controlador POX nos permitió la configuración de políticas específicas para el manejo de tráfico de red. El uso de la herramienta nos facilitó la incorporación de un módulo personalizado para actuar como firewall, utilizando su configuración para interceptar y descartar paquetes según las reglas definidas.

2.4. Firewall

El módulo de firewall implementado en este trabajo está diseñado para interactuar con el controlador POX y administrar políticas de red definidas a través de un archivo JSON. Este módulo permite interceptar el tráfico en los switches administrados y aplicar reglas que descartan paquetes según los criterios establecidos en dichas políticas. El enfoque principal fue la modularidad y la flexibilidad, logradas mediante la carga dinámica de configuraciones externas y la capacidad de generar reglas de manera programática.

La configuración del firewall se define en el archivo firewall_policies.json. Este archivo contiene dos elementos fundamentales: el identificador del switch donde se aplicarán las políticas, bajo la clave "firewall_switch_id", y una lista de políticas específicas dentro de la clave "discard_policies". Cada política especifica condiciones bajo las cuales los paquetes deben ser descartados, utilizando campos como el puerto de destino ("tp_dst"), el protocolo de red ("nw_proto"), la dirección de origen ("nw_src"), la dirección de destino ("nw_dst"), el tipo de datagrama ("dl_type"), la dirección MAC de origen ("dl_src") y la dirección MAC de destino ("dl_dst"). Por ejemplo, una regla como:

```
{ "tp_dst": "80" }
```

indica que todo el tráfico dirigido al puerto 80 debe ser bloqueado, mientras que una regla más específica como la siguiente, descarta únicamente paquetes UDP provenientes de la dirección 10.0.0.1 con destino al puerto 5001 en redes IPv4:

```
{ "tp_dst": "5001", "nw_proto": "udp", "nw_src": "10.0.0.1", "dl_type": "ipv4" }
```

La lógica del módulo se basa en el método load_policies, que carga estas configuraciones al inicializar el controlador. Una vez establecida la conexión entre el controlador y el switch identificado, el método _handle_ConnectionUp aplica las reglas al switch utilizando el protocolo OpenFlow. Cada regla se genera mediante el método _rule_from_policy, que traduce los campos del JSON a atributos de las clases de OpenFlow, como of.ofp_flow_mod.

Un aspecto destacado de la implementación es la generación automática de variantes de políticas. Cuando una regla no especifica ciertos campos requeridos como el tipo de datagrama o el protocolo de red, el módulo crea versiones adicionales para cubrir todos los casos posibles. Por ejemplo, si la política en el archivo no especifica el protocolo de transporte, se generan automáticamente las reglas necesarias para bloquear todos los protocolos de transporte (TCP, UDP, etc.). En cambio, si se especifica un protocolo de transporte en particular tan solo se descartarán los paquetes con ese protocolo en particular.

Esta medida simplifica la escritura de reglas, y refeleja de una forma minimal las politicas aplicadas.

3. Pruebas

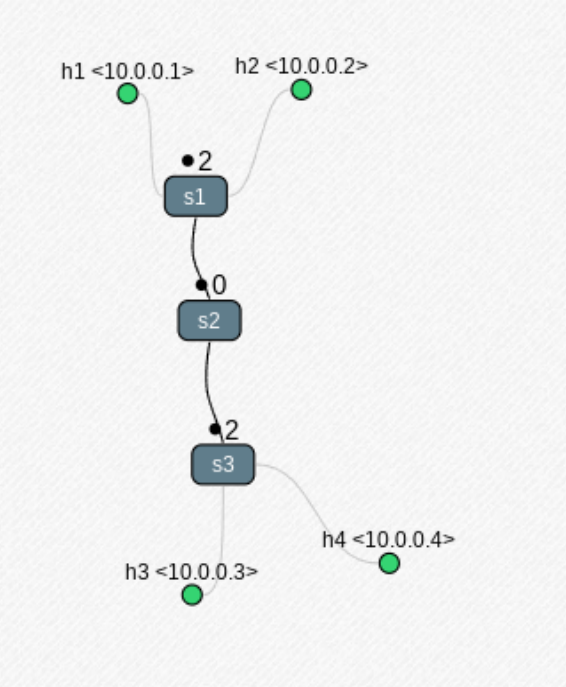
3.1. Configuración Inicial

Para probar el correcto funcionamiento del proyecto, iniciamos todos los componentes del sistema (POX con firewall y mininet).

3.1.1. Topología

Nuestra topología utiliza fue la descripta anterirmente, con 3 switches:

- s1: que se conecta con h1, h2 y s2
- s2: que se conecta s1 con s3
- s3: que se conecta con h3, h4 y s2



3.1.2. Políticas

Las reglas se establecieron el fin de cubrir los siguientes casos:

1. Se deben descartar todos los mensajes cuyo puerto destino sea 80.
2. Se deben descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y estén utilizando el protocolo UDP.
3. Se deben elegir dos hosts cualesquiera y los mismos no deben poder comunicarse de ninguna forma.

Es por ello que determinamos las siguientes políticas en el archivo JSON.

```
{
    "tp_dst": "80",
},
{
    "tp_dst": "5001",
    "nw_proto": "udp",
    "dl_src": "00:00:00:00:00:01"
},
{
    "dl_src": "00:00:00:00:00:02",
    "dl_dst": "00:00:00:00:00:03"
},
{
    "dl_src": "00:00:00:00:00:03",
    "dl_dst": "00:00:00:00:00:02"
}
```

Estas políticas determinan el siguiente clasificador:

Rule	Dst Port	Transport layer Protocol	Src MAC Addr	Dst MAC Addr	Action
R_1	80	tcp	*	*	Deny
R_2	80	udp	*	*	Deny
R_3	5001	udp	00:00:00:00:00:01	*	Deny
R_4	*	*	00:00:00:00:00:02	00:00:00:00:00:03	Deny
R_5	*	*	00:00:00:00:00:03	00:00:00:00:00:02	Deny

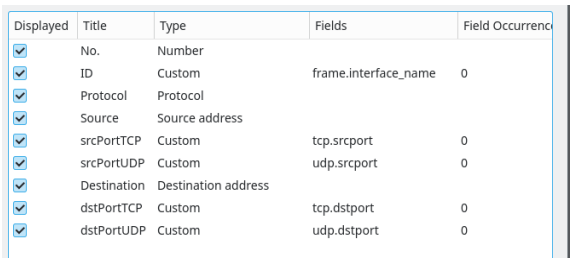
Decidimos usar a h2 y h3 como host que no deben poder comunicarse para cumplir con política número 3.

También pre establecimos las MAC Address de los hosts y no solo las IPs para asegurarnos que están no tomen un valor aleatorio y que nos permitan filtrar de forma más contundente todos los paquetes que intenten viajar a un endpoint en particular.

Por otro lado, dado la topología, el tráfico entre h3 y h4 no sera mediado y por ende no le aplicarán las politicas ya que solo necesitan del s3 para contactarse y este no tiene las reglas establecidas.

3.1.3. Configuración de Wireshark

Para atrapar los paquetes y visualizar los campos relevantes, escuchamos todas las interfaces de los switches de la topología y configuramos las siguientes columnas.



Displayed	Title	Type	Fields	Field Occurrenc
☑	No.	Number		
☑	ID	Custom	frame.interface_name	0
☑	Protocol	Protocol		
☑	Source	Source address		
☑	srcPortTCP	Custom	tcp.srcport	0
☑	srcPortUDP	Custom	udp.srcport	0
☑	Destination	Destination address		
☑	dstPortTCP	Custom	tcp.dstport	0
☑	dstPortUDP	Custom	udp.dstport	0

3.1.4. Comandos iniciales

Iniciamos primero POX con nuestra implementación del Firewall y las reglas establecidas en el archivo JSON. Luego levantamos mininet y el controlador de s1 automaticamente aplica las politicas de red.

```
python3 ./pox.py forwarding.l2_learning firewall

sudo mn --custom ./topology.py --topo customTopo,switches=3 --arp
--switch ovsk --controller remote
```

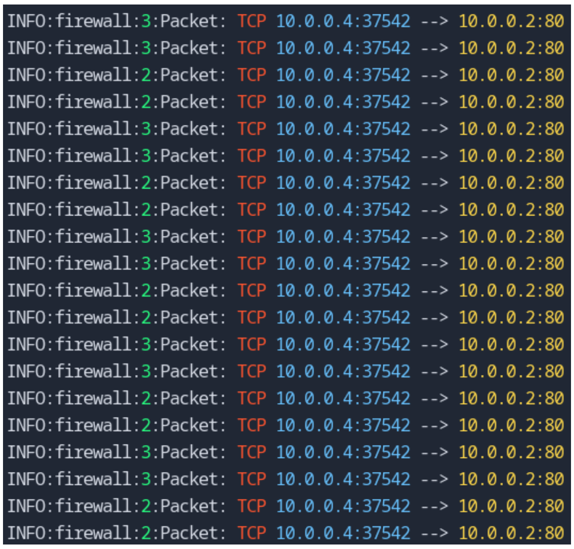
3.2. Pruebas

3.2.1. Conexión fallida hacia puerto 80

```
mininet> h4 curl h2
```

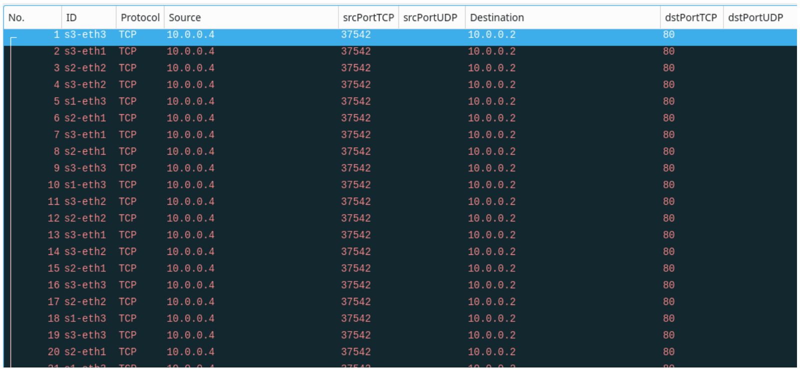
Hacemos unas consulta HTTP (puerto 80) entre h4 y h2 (interceptado por el firewall).

En los logs del controlador podemos ver que nunca se levanta el paquete en el s1 porque el mismo lo descarta, pero si lo hace en los otros 2 switches



```
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:3:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
INFO:firewall:2:Packet: TCP 10.0.0.4:37542 --> 10.0.0.2:80
```

En Wireshark en cambio, podemos ver que el paquete entra por la interfaz de s1 pero no sale por la otra hacia h2 ya que fue descartado por la política del puerto 80.



No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
1	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
2	s3-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
3	s2-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
4	s3-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
5	s1-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
6	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
7	s3-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
8	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
9	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
10	s1-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
11	s3-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
12	s2-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
13	s3-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
14	s3-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
15	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
16	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
17	s2-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
18	s1-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
19	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
20	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	

3.2.2. Cliente-servidor por puerto 80 sobre UDP

```
mininet> h2 iperf -u -s -p 80&
mininet> h4 iperf -u -c h2 -p 80
```

Ponemos a h2 a escuchar a modo de ”servidor”sobre el puerto 80. Enviamos una solicitud UDP desde h4 consultando por el ancho de banda. Mismo, caso que en la prueba anterior como trabajamos sobre el puerto 80, los paquetes serán descartados.

```
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:3:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
INFO:firewall:2:Packet: UDP 10.0.0.4:42802 --> 10.0.0.2:80
```

```
mininet> h4 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 10.0.0.4 port 42802 connected with 10.0.0.2 port 80
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0160 sec  1.25 MBytes  1.05 Mbits/sec
[  1] Sent 896 datagrams
[  3] WARNING: did not receive ack of last datagram after 10 tries.
```

No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
219	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
220	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
221	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
222	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
223	s1-eth3	UDP	10.0.0.4		42802	10.0.0.2		80
224	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
225	s1-eth3	UDP	10.0.0.4		42802	10.0.0.2		80
226	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
227	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
228	s2-eth2	UDP	10.0.0.4		42802	10.0.0.2		80
229	s1-eth3	UDP	10.0.0.4		42802	10.0.0.2		80
230	s1-eth3	UDP	10.0.0.4		42802	10.0.0.2		80
231	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
232	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
233	s2-eth2	UDP	10.0.0.4		42802	10.0.0.2		80
234	s1-eth3	UDP	10.0.0.4		42802	10.0.0.2		80
235	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80
236	s2-eth2	UDP	10.0.0.4		42802	10.0.0.2		80
237	s1-eth3	UDP	10.0.0.4		42802	10.0.0.2		80
238	s2-eth1	UDP	10.0.0.4		42802	10.0.0.2		80

3.2.3. Conexión UDP fallida sobre el puerto 5001

```
mininet> h4 iperf -u -s -p 5001&
mininet> h1 iperf -u -c h4 -p 5001
```

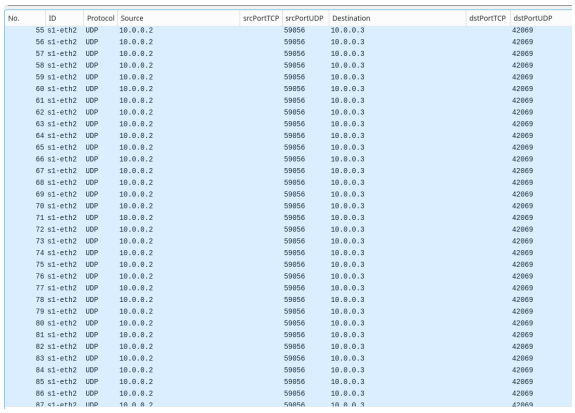
Se prepara de la misma forma que el caso anterior, pero usando el h4 como servidor y h1 como cliente, hacia el puerto 5001. Tal como se establece en las políticas esta situación debe ser denegada.

```
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 35570 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.
```

En los logs del controlador, no figura nada dado que son descartados al ingresar a s1.

```
python3 ./pox.py forwarding.l2_learning firewall
POX 0.9.0 (ichthyosaur) / Copyright 2011-2023 James McCauley, e
t al.
INFO:firewall:Enabling Firewall Module
INFO:core:POX 0.9.0 (ichthyosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:firewall:Firewall rules installed on 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
```

Podemos ver en Wireshark que todos los paquetes figuran en la misma interfaz de s1, donde se descarta.



No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
55	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
56	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
57	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
58	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
59	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
60	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
61	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
62	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
63	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
64	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
65	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
66	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
67	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
68	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
69	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
70	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
71	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
72	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
73	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
74	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
75	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
76	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
77	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
78	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
79	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
80	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
81	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
82	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
83	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
84	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
85	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
86	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069
87	s1-eth2	UDP	10.0.0.2		59856	10.0.0.3		42069

3.2.4. Conectividad entre h2 y h3

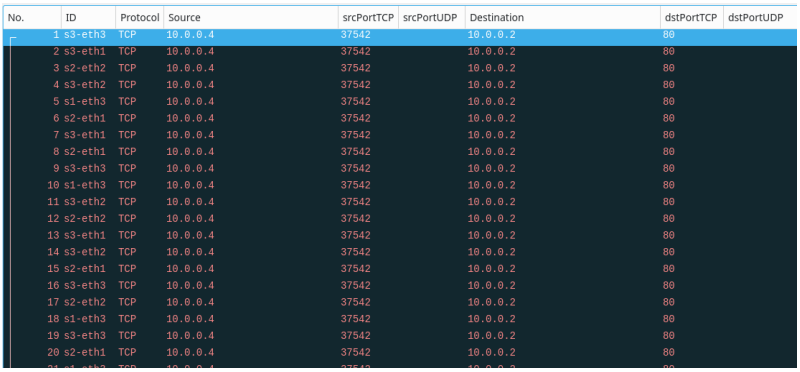
La última establece que no debe haber conexión de ningún tipo entre h2 y h3. En esta prueba probaremos usar a h2 como origen y h3 como destino.

```
mininet> h2 ping h3
```

```
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
43 packets transmitted, 0 received, 100% packet loss, time 43003ms
```

Al igual que en el caso anterior el s1 descarta el paquete y en el caso del controlador no lo loguea pero en Wireshark se ve como no sale por otra interfaz distinta a la que llego.

```
python3 ./pox.py forwarding.l2_learning firewall
POX 0.9.0 (ichthyosaur) / Copyright 2011-2023 James McCauley, e
t al.
INFO:firewall:Enabling Firewall Module
INFO:core:POX 0.9.0 (ichthyosaur) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:firewall:Firewall rules installed on 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
```

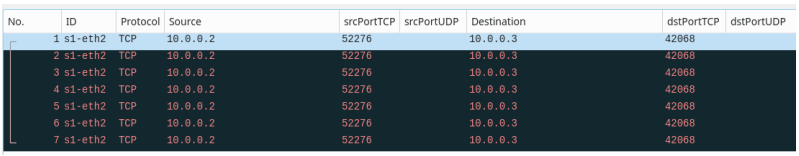


No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
1	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
2	s3-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
3	s2-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
4	s3-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
5	s1-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
6	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
7	s3-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
8	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
9	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
10	s1-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
11	s3-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
12	s2-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
13	s3-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
14	s3-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
15	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	
16	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
17	s2-eth2	TCP	10.0.0.4	37542		10.0.0.2	80	
18	s1-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
19	s3-eth3	TCP	10.0.0.4	37542		10.0.0.2	80	
20	s2-eth1	TCP	10.0.0.4	37542		10.0.0.2	80	

Probamos lo mismo, pero usando TCP.

```
mininet> h3 iperf -s -p 42068&
mininet> h2 iperf -c h3 -p 42068
```

```
mininet> h3 iperf -s -p 42068&
------------------------------------------------------------
Server listening on TCP port 42069
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
mininet> h2 iperf -c h3 -p 42068
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 42068
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
tcp connect failed: Connection timed out
[  1] local 0.0.0.0 port 0 connected with 10.0.0.3 port 42068
```



No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
1	s1-eth2	TCP	10.0.0.2	52276		10.0.0.3	42068	
2	s1-eth2	TCP	10.0.0.2	52276		10.0.0.3	42068	
3	s1-eth2	TCP	10.0.0.2	52276		10.0.0.3	42068	
4	s1-eth2	TCP	10.0.0.2	52276		10.0.0.3	42068	
5	s1-eth2	TCP	10.0.0.2	52276		10.0.0.3	42068	
6	s1-eth2	TCP	10.0.0.2	52276		10.0.0.3	42068	
7	s1-eth2	TCP	10.0.0.2	52276		10.0.0.3	42068	

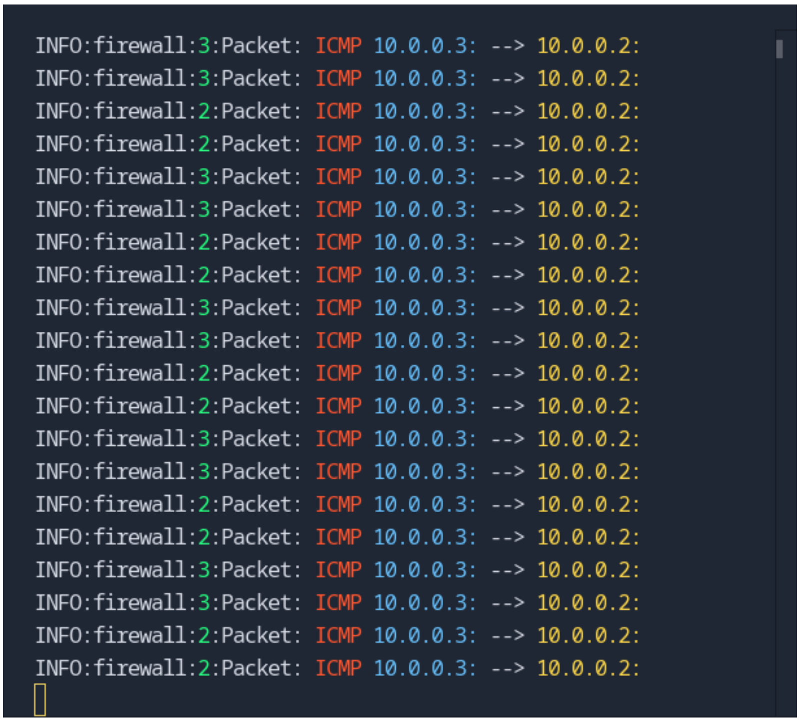
3.2.5. Conectividad entre h3 y h2

En este caso comprobaremos el sentido opuesto, h3 como origen y h2 como destino.

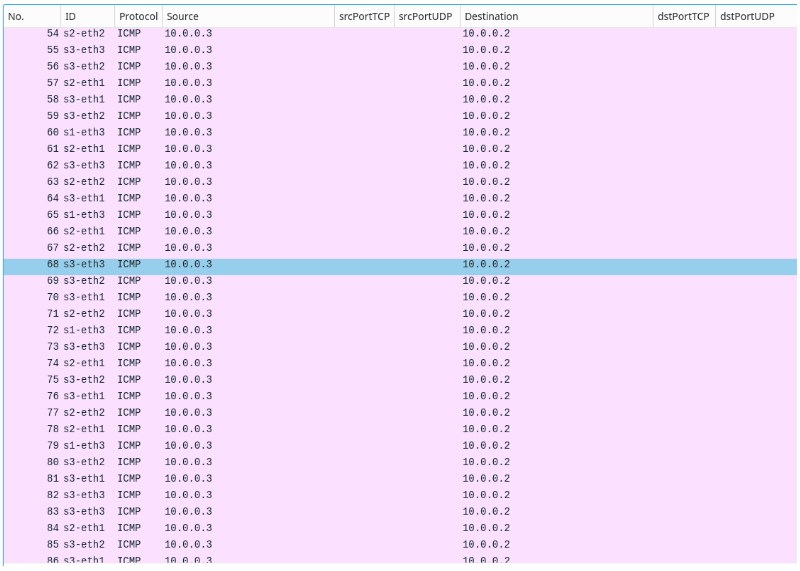
```
mininet> h3 ping h2
```

```
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5102ms
```

En este caso, si se llega a ver más movimiento de paquetes tanto en el controlador como en Wireshark, ya que los paquetes viajan exitosamente por s3 y s2, no así por s1, donde terminan siendo descartados.



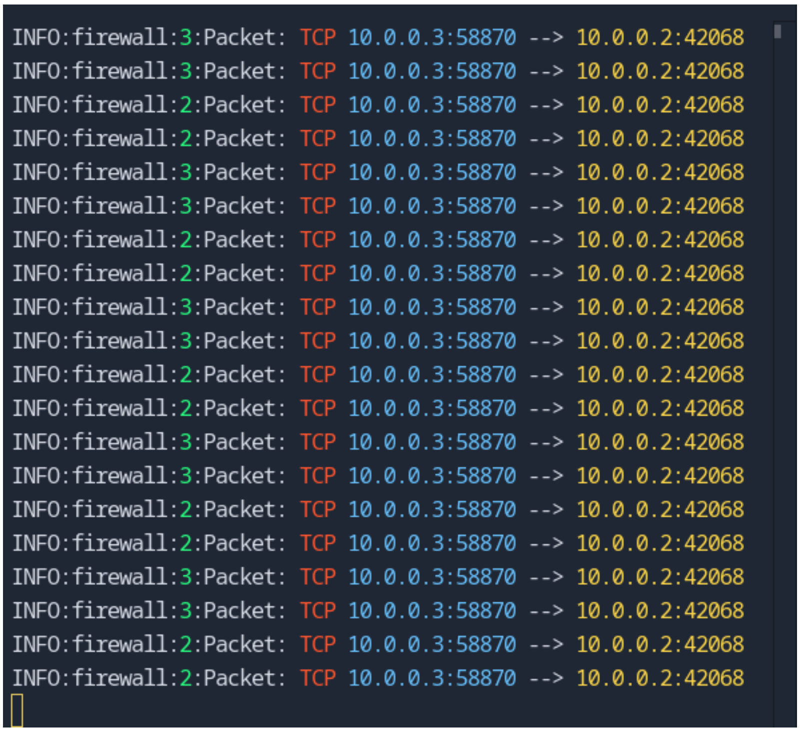
```
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:3:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
INFO:firewall:2:Packet: ICMP 10.0.0.3: --> 10.0.0.2:
```

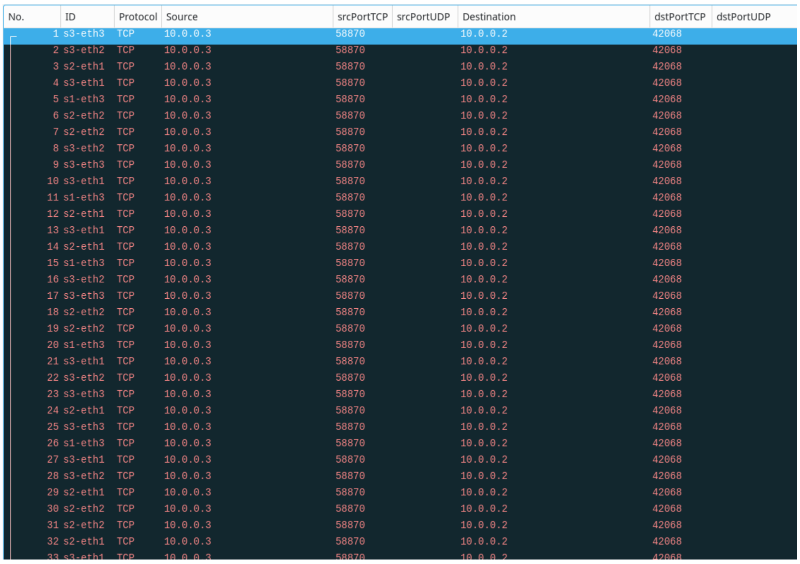


No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
54	s2-eth2	ICMP	10.0.0.3			10.0.0.2		
55	s3-eth3	ICMP	10.0.0.3			10.0.0.2		
56	s3-eth2	ICMP	10.0.0.3			10.0.0.2		
57	s2-eth1	ICMP	10.0.0.3			10.0.0.2		
58	s3-eth1	ICMP	10.0.0.3			10.0.0.2		
59	s3-eth2	ICMP	10.0.0.3			10.0.0.2		
60	s1-eth3	ICMP	10.0.0.3			10.0.0.2		
61	s2-eth1	ICMP	10.0.0.3			10.0.0.2		
62	s3-eth3	ICMP	10.0.0.3			10.0.0.2		
63	s2-eth2	ICMP	10.0.0.3			10.0.0.2		
64	s3-eth1	ICMP	10.0.0.3			10.0.0.2		
65	s1-eth3	ICMP	10.0.0.3			10.0.0.2		
66	s2-eth1	ICMP	10.0.0.3			10.0.0.2		
67	s2-eth2	ICMP	10.0.0.3			10.0.0.2		
68	s3-eth3	ICMP	10.0.0.3			10.0.0.2		
69	s3-eth2	ICMP	10.0.0.3			10.0.0.2		
70	s3-eth1	ICMP	10.0.0.3			10.0.0.2		
71	s2-eth2	ICMP	10.0.0.3			10.0.0.2		
72	s1-eth3	ICMP	10.0.0.3			10.0.0.2		
73	s3-eth3	ICMP	10.0.0.3			10.0.0.2		
74	s2-eth1	ICMP	10.0.0.3			10.0.0.2		
75	s3-eth2	ICMP	10.0.0.3			10.0.0.2		
76	s3-eth1	ICMP	10.0.0.3			10.0.0.2		
77	s2-eth2	ICMP	10.0.0.3			10.0.0.2		
78	s2-eth1	ICMP	10.0.0.3			10.0.0.2		
79	s1-eth3	ICMP	10.0.0.3			10.0.0.2		
80	s3-eth2	ICMP	10.0.0.3			10.0.0.2		
81	s3-eth1	ICMP	10.0.0.3			10.0.0.2		
82	s3-eth3	ICMP	10.0.0.3			10.0.0.2		
83	s3-eth3	ICMP	10.0.0.3			10.0.0.2		
84	s2-eth1	ICMP	10.0.0.3			10.0.0.2		
85	s3-eth2	ICMP	10.0.0.3			10.0.0.2		
86	s3-eth1	ICMP	10.0.0.3			10.0.0.2		

Probamos los mismo con TCP.

```
mininet> h2 iperf -s -p 42068&
mininet> h3 iperf -c h2 -p 42068
```





No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
1	s3-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
2	s3-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
3	s2-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
4	s3-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
5	s1-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
6	s2-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
7	s2-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
8	s3-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
9	s3-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
10	s3-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
11	s1-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
12	s2-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
13	s3-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
14	s2-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
15	s1-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
16	s3-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
17	s3-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
18	s2-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
19	s2-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
20	s1-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
21	s3-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
22	s3-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
23	s3-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
24	s2-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
25	s3-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
26	s1-eth3	TCP	10.0.0.3	58870		10.0.0.2	42068	
27	s3-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
28	s3-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
29	s2-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
30	s2-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
31	s2-eth2	TCP	10.0.0.3	58870		10.0.0.2	42068	
32	s2-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	
33	s3-eth1	TCP	10.0.0.3	58870		10.0.0.2	42068	

Y probamos lo mismo con UDP.

```
mininet> h2 iperf -u -s -p 42069&
mininet> h3 iperf -u -c h2 -p 42069
```

```
mininet> h2 iperf -u -s -p 42069&
------------------------------------------------------------
Server listening on TCP port 42068
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
mininet> h3 iperf -u -c h2 -p 42069
------------------------------------------------------------
Client connecting to 10.0.0.2, UDP port 42069
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.3 port 36834 connected with 10.0.0.2 port 42069
[ 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.
```

```
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:3:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
INFO:firewall:2:Packet: UDP 10.0.0.3:36834 --> 10.0.0.2:42069
```

No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
45	s2-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
46	s1-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
47	s2-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
48	s2-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
49	s2-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
50	s3-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
51	s3-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
52	s3-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
53	s3-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
54	s3-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
55	s3-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
56	s3-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
57	s3-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
58	s3-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
59	s3-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
60	s3-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
61	s3-eth1	UDP	10.0.0.3		36834	10.0.0.2		42069
62	s3-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
63	s3-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
64	s2-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
65	s3-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
66	s2-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
67	s1-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
68	s1-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
69	s2-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
70	s2-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
71	s1-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
72	s1-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
73	s2-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
74	s3-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069
75	s3-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
76	s3-eth2	UDP	10.0.0.3		36834	10.0.0.2		42069
77	s1-eth3	UDP	10.0.0.3		36834	10.0.0.2		42069

3.2.6. Casos permitidos

Para corroborar el correcto funcionamiento del Firewall es decir que tan solo bloque aquellos paquetes dictados por las políticas, realizamos una serie de ejemplos que ilustran el correcto funcionamiento del midlebox.

PING ALL

En esta caso se puede ver que, menos los dos puertos que elegimos para que estén bloqueados entre si(h2 y h3), lo host se pueden comunicar sin ningún inconveniente.

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 X h4
h3 -> h1 X h4
h4 -> h1 h2 h3
*** Results: 16% dropped (10/12 received)
```

POLITICA 2 CON PROTOCOLO TCP

En este caso podemos ver como efectivamente en los logs del controlador los paquetes pasan a través el switch 1 sin ningún inconveniente, y en el caso de WireSharck como los paquetes entran por una interfaz del switch 1 y salen por otra.

```
mininet> h4 iperf -s -p 5001&
mininet> h1 iperf -c h4 -p 5001
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.1 port 34060 connected with 10.0.0.4 port 5001 (icwnd/mss/irt
t=14/1448/5894)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0011 sec  6.63 GBytes  5.69 Gbits/sec
```

```
INFO:firewall:3:Packet: TCP 10.0.0.4:5001 --> 10.0.0.1:34060
INFO:firewall:3:Packet: TCP 10.0.0.4:5001 --> 10.0.0.1:34060
INFO:firewall:2:Packet: TCP 10.0.0.4:5001 --> 10.0.0.1:34060
INFO:firewall:2:Packet: TCP 10.0.0.4:5001 --> 10.0.0.1:34060
INFO:firewall:1:Packet: TCP 10.0.0.4:5001 --> 10.0.0.1:34060
INFO:firewall:1:Packet: TCP 10.0.0.4:5001 --> 10.0.0.1:34060
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
INFO:firewall:1:Packet: TCP 10.0.0.1:34060 --> 10.0.0.4:5001
```

No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
1	s1-eth1	TCP	10.0.0.1	34060		10.0.0.4	5001	
2	s1-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
3	s3-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
4	s1-eth3	TCP	10.0.0.4	5001		10.0.0.1	34060	
5	s3-eth3	TCP	10.0.0.4	5001		10.0.0.1	34060	
6	s1-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
7	s3-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
8	s1-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
9	s3-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
10	s1-eth3	TCP	10.0.0.4	5001		10.0.0.1	34060	
11	s3-eth3	TCP	10.0.0.4	5001		10.0.0.1	34060	
12	s1-eth3	TCP	10.0.0.4	5001		10.0.0.1	34060	
13	s3-eth3	TCP	10.0.0.4	5001		10.0.0.1	34060	
14	s1-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
15	s3-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
16	s1-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
17	s3-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
18	s1-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
19	s3-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
20	s1-eth3	TCP	10.0.0.1	34060		10.0.0.4	5001	
21	s2-eth1	TCP	10.0.0.1	34060		10.0.0.4	5001	
22	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
23	s3-eth2	TCP	10.0.0.4	5001		10.0.0.1	34060	
24	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
25	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
26	s3-eth2	TCP	10.0.0.4	5001		10.0.0.1	34060	
27	s3-eth2	TCP	10.0.0.4	5001		10.0.0.1	34060	
28	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
29	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
30	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
31	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
32	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	
33	s3-eth2	TCP	10.0.0.1	34060		10.0.0.4	5001	

PUERTO 443 (No 80)

En este caso podemos ver la situación en donde se envían información a través del h1 por UDP a h4 al puerto 433 (no al 80) y los paquetes pasan a través del Firewall.

```
mininet> h1 iperf -u -c h4 -p 443
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 443
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.1 port 56800 connected with 10.0.0.4 port 443
[ 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] 0.0000-10.0109 sec  1.25 MBytes  1.05 Mbits/sec   0.015 ms 0/895 (0%)
```

No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
5354	s1-eth1	UDP	10.0.0.1		56800	10.0.0.4		443
5355	s1-eth1	UDP	10.0.0.1		56800	10.0.0.4		443
5356	s1-eth1	UDP	10.0.0.1		56800	10.0.0.4		443
5357	s1-eth1	UDP	10.0.0.1		56800	10.0.0.4		443
5358	s1-eth1	UDP	10.0.0.1		56800	10.0.0.4		443
5359	s1-eth1	UDP	10.0.0.4		443	10.0.0.1		56800
5360	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5361	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5362	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5363	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5364	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5365	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5366	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5367	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5368	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5369	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5370	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5371	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5372	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5373	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5374	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5375	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5376	s2-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5377	s2-eth2	UDP	10.0.0.4		443	10.0.0.1		56800
5378	s3-eth2	UDP	10.0.0.1		56800	10.0.0.4		443
5379	s3-eth2	UDP	10.0.0.4		443	10.0.0.1		56800

```
INFO:firewall:1:Packet: UDP 10.0.0.1:56800 --> 10.0.0.4:443
INFO:firewall:1:Packet: UDP 10.0.0.1:56800 --> 10.0.0.4:443
INFO:firewall:2:Packet: UDP 10.0.0.1:56800 --> 10.0.0.4:443
INFO:firewall:2:Packet: UDP 10.0.0.1:56800 --> 10.0.0.4:443
INFO:firewall:3:Packet: UDP 10.0.0.1:56800 --> 10.0.0.4:443
INFO:firewall:3:Packet: UDP 10.0.0.1:56800 --> 10.0.0.4:443
INFO:firewall:3:Packet: UDP 10.0.0.4:443 --> 10.0.0.1:56800
INFO:firewall:3:Packet: UDP 10.0.0.4:443 --> 10.0.0.1:56800
INFO:firewall:2:Packet: UDP 10.0.0.4:443 --> 10.0.0.1:56800
INFO:firewall:2:Packet: UDP 10.0.0.4:443 --> 10.0.0.1:56800
INFO:firewall:1:Packet: UDP 10.0.0.4:443 --> 10.0.0.1:56800
INFO:firewall:1:Packet: UDP 10.0.0.4:443 --> 10.0.0.1:56800
```

3.2.7. Casos no contemplados

Como mencionamos anteriormente, la conexión entre h3 y h4 existe sin un firewall en su camino por lo que podemos establecer una conexión vía puerto 80 sin restricción alguna.

```
mininet> h4 iperf -s -p 80&
mininet> h3 iperf -c h3 -p 80
```

```
mininet> h3 iperf -c h4 -p 80
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 80
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.3 port 37636 connected with 10.0.0.4 port 80 (icwnd/mss/irtt=1
4/1448/2132)
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-10.0005 sec  9.82 GBytes  8.43 Gbits/sec
```

En este caso podemos ver que la conexión es exitosa y no se dropean paquetes en el camino ya que solo pasan por interfaces relacionadas al s3.

```
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:34806
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:34806
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:34806
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:34806
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:34806
INFO:firewall:3:Packet: TCP 10.0.0.3:37636 --> 10.0.0.4:80
INFO:firewall:3:Packet: TCP 10.0.0.3:37636 --> 10.0.0.4:80
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
INFO:firewall:3:Packet: TCP 10.0.0.4:80 --> 10.0.0.3:37636
```

No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
1	s3-eth1	TCP	10.0.0.3	37636		10.0.0.4	80	
2	s3-eth1	TCP	10.0.0.4	80		10.0.0.3	37636	
3	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
4	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
5	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
6	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
7	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
8	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
9	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
10	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
11	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
12	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
13	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
14	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
15	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
16	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
17	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
18	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
19	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
20	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
21	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
22	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
23	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
24	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
25	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
26	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
27	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
28	s3-eth2	TCP	10.0.0.4	80		10.0.0.3	37636	
29	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
30	s3-eth2	TCP	10.0.0.3	37636		10.0.0.4	80	
31	s3-eth1	TCP	10.0.0.3	37636		10.0.0.4	80	
32	s3-eth1	TCP	10.0.0.3	37636		10.0.0.4	80	
33	s3-eth1	TCP	10.0.0.3	37636		10.0.0.4	80	

La misma situación sucede en el caso de que estemos utilizando el protocolo UDP.

```
mininet> h3 iperf -u -c h4 -p 80
------------------------------------------------------------
Client connecting to 10.0.0.4, UDP port 80
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  1] local 10.0.0.3 port 44339 connected with 10.0.0.4 port 80
[ 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] 0.0000-10.0140 sec  1.25 MBytes  1.05 Mbits/sec   0.011 ms 0/895 (0%)
```

```
INFO:firewall:3:Packet: UDP 10.0.0.3:44339 --> 10.0.0.4:80
INFO:firewall:3:Packet: UDP 10.0.0.3:44339 --> 10.0.0.4:80
INFO:firewall:3:Packet: UDP 10.0.0.4:80 --> 10.0.0.3:44339
INFO:firewall:3:Packet: UDP 10.0.0.4:80 --> 10.0.0.3:44339
```

No.	ID	Protocol	Source	srcPortTCP	srcPortUDP	Destination	dstPortTCP	dstPortUDP
1	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
2	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
3	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
4	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
5	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
6	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
7	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
8	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
9	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
10	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
11	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
12	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80
13	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
14	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
15	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
16	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
17	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
18	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
19	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
20	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
21	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
22	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
23	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
24	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
25	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
26	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
27	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
28	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
29	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
30	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
31	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
32	s3-eth2	UDP	10.0.0.3		44339	10.0.0.4		80
33	s3-eth1	UDP	10.0.0.3		44339	10.0.0.4		80

4. Preguntas

1. ¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?

   La diferencia entre un switch y un router es fundamental. Un switch opera principalmente en la capa de enlace de datos del modelo TCP/IP, utilizando direcciones MAC para identificar los dispositivos dentro de la misma red local. Su función es facilitar el intercambio de datos entre dispositivos que están conectados directamente y dentro de un mismo dominio, asegurando que los datos se envíen únicamente al dispositivo de destino correcto dentro de la LAN. El switch crea una tabla de direcciones MAC, de modo que puede reenviar los paquetes a través del puerto correspondiente a la dirección de destino. De esta manera, el switch optimiza el flujo de datos dentro de la red local sin intervenir en el enrutamiento a redes externas.

   Por otro lado, un router funciona más en la capa de red y se enfoca en la administración y redirección del tráfico entre diferentes redes. Utilizando direcciones IP, un router determina la ruta óptima para que los paquetes lleguen a su destino, incluso si este se encuentra en una red externa o remota. Mientras que el switch es responsable de las conexiones dentro de la misma red local, el router es clave para conectar distintas redes y dirigir el tráfico hacia otras redes a través de un proceso llamado enrutamiento. Este proceso permite que el router consulte una tabla de enrutamiento, donde se almacenan rutas posibles hacia redes externas, y elija el camino más eficiente para cada paquete.

   Ambos dispositivos comparten la función de dirigir el tráfico de red, pero en distintos contextos. Mientras el switch conecta dispositivos dentro de la misma red y facilita la comunicación local, el router permite la comunicación entre redes separadas y administra el flujo de tráfico hacia y desde la red interna a redes externas. En implementaciones modernas, tanto los switches como los routers pueden contar con capacidades avanzadas de configuración y administración, como ocurre en las Redes Definidas por Software (SDN), donde ambos pueden ser controlados mediante software para adaptar y optimizar la red.

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

   La diferencia entre un switch convencional y un switch OpenFlow radica en el nivel de control y flexibilidad que cada uno ofrece en la administración de tráfico de red.

   Un switch convencional se basa en protocolos y algoritmos preestablecidos para la redirección de paquetes dentro de una red. Opera de forma autónoma, gestionando el tráfico según direcciones MAC y manteniendo una tabla para reenviar los paquetes hacia el puerto correcto. Aunque permite cierto nivel de configuración, un switch convencional sigue decisiones de enrutamiento y reenvío fijas, establecidas por el fabricante, lo que limita la capacidad de realizar ajustes personalizados o intervenir en la lógica de conmutación sin depender de hardware específico.

   Por otro lado, un switch OpenFlow es un tipo de switch programable que forma parte de la arquitectura de Redes SDN. Este tipo de switch permite un control mucho más detallado y particularmente dinámico sobre el flujo de datos en la red, al separar el plano de control del plano de datos. A diferencia del switch común, el switch OpenFlow puede ser gestionado desde un controlador central que envía instrucciones de reenvío específicas para cada flujo de paquetes, usando el protocolo OpenFlow. Esto significa que, en lugar de depender de las decisiones internas del hardware, el switch OpenFlow ejecuta políticas definidas en tiempo real por el controlador, permitiendo mayor flexibilidad en la implementación de reglas de seguridad, control de tráfico y adaptación de la red. A su vez, como es en el caso de este trabajo, permite dinámicamente agregar opciones de seguridad por medio de un firewall, como bloquear direcciones o evitar comunicación entre hosts.

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

   No es posible reemplazar todos los routers de Internet por switches OpenFlow, especialmente en un escenario inter-AS (sistema autónomo), debido a las diferencias funcionales y a los requisitos específicos del enrutamiento entre sistemas autónomos. En el contexto de Internet,

los routers interAS cumplen una función fundamental en el enrutamiento a gran escala, facilitando la comunicación entre redes que pertenecen a distintos proveedores o entidades autónomas. Este tipo de enrutamiento se basa en protocolos como BGP (Border Gateway Protocol), los cuales están diseñados para manejar políticas complejas y rutas optimizadas que satisfagan acuerdos específicos entre sistemas autónomos, algo que no es compatible con la infraestructura y diseño de los switches OpenFlow.

Mientras que los switches OpenFlow pueden programarse para gestionar el tráfico a nivel de flujo con una flexibilidad considerable dentro de un sistema autónomo (intra-AS), su arquitectura no está diseñada para soportar la dinámica y las políticas específicas del enrutamiento entre distintos sistemas autónomos. Los routers inter-AS no solo toman decisiones de enrutamiento basadas en la mejor ruta para cada paquete, sino que también aplican políticas de acceso, priorización y distribución de cargas a nivel global, como es el caso de BGP. Estos son elementos fundamentales para mantener la estabilidad y eficiencia de Internet. Además, los routers tradicionales están optimizados para manejar tablas de enrutamiento extensas y complejas. Estas características exceden las capacidades de conmutación y control de flujo de los switches OpenFlow.

5. Dificultades

En esta sección se detallan algunas de las dificultades que se enfrentaron durante el desarrollo de este trabajo y el estudio de sus temáticas.

- La principal dificultad encontrada al realizar este trabajo, definitivamente fue familiarizarnos con el uso de las herramientas requeridas para su desarrollo. Se necesitó armar un entorno con múltiples librerías (Pox, Mininet, Wireshark, etc.) que interactuaran entre sí. Por ejemplo, el código que compone nuestro firewall es relativamente simple, de poca longitud y, sin embargo, para llevar a cabo su implementación fue necesario comprender muy a fondo el funcionamiento de Pox y como configurar las distintas variables en juego. Una vez superada esta dificultad, el posterior desarrollo de este firewall no propuso muchas complicaciones.
- Por otro lado, el software de Mininet fue, nuevamente, complicado de inicializar. En una primera instancia nos encontramos con que nuestra topología no lograba comunicar a los Hosts entre sí y se tardó un rato largo entender el problema que estábamos teniendo. Una vez solucionado este problema, el desarrollo de la topología dinámica no propuso desafíos posteriores significativos.

6. Conclusión

A lo largo del desarrollo de este trabajo práctico, hemos implementado y explorado un caso aplicado de Software Defined Networks (SDN), utilizando OpenFlow como protocolo de comunicación. Este enfoque nos permitió diseñar y construir una red configurable con una cantidad variable de switches, haciendo uso de Mininet con un conocimiento más avanzado respecto a experiencias previas.

Además, incorporamos el uso del controlador POX y creamos exitosamente nuestro Firewall, a través del cual definimos políticas específicas para gestionar el tráfico de red, incluyendo el bloqueo de casos particulares. Estas políticas fueron diseñadas de manera parametrizable y almacenadas en un archivo externo, permitiendo que un administrador de red pueda modificarlas sin necesidad de alterar el código fuente, mejorando así la flexibilidad y la administración de la solución.

Por último, profundizamos en el uso de Wireshark, configurando sesiones de captura propias, personalizando columnas, y capturando el tráfico en todas las interfaces de los switches de la topología. Esto nos permitió analizar en detalle el recorrido de los paquetes, identificando con precisión los puntos donde estos eran permitidos o bloqueados.