

# Trabajo Práctico 2 — Software-Defined Networks

Fecha de entrega: 21/11/23

[75.43] Introducción a los Sistemas Distribuidos  
Segundo Cuatrimestre de 2023

## Integrantes Grupo B11

Alumno	Padrón
DUCA, Francisco	106308
GOMEZ, Nahuel	106514
SCHIFFER, Brandon	107890
SCAZZOLA, Martín	106403
SANTANDER, Valentín	105637

## Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Hipótesis/Supuestos</b>	<b>2</b>
<b>3. Detalles de implementación</b>	<b>3</b>
3.1. Topología . . . . .	3
3.2. Firewall . . . . .	3
<b>4. Pruebas</b>	<b>4</b>
4.1. Resultado de las simulaciones . . . . .	4
4.2. Prueba Pingall . . . . .	5
4.3. Firewall - Aplicación de reglas . . . . .	8
4.4. l2 learning . . . . .	9
4.5. Conexión exitosa . . . . .	10
4.5.1. UDP . . . . .	10
4.5.2. TCP . . . . .	11
4.6. Bloqueo TCP puerto 80 . . . . .	12
4.7. Bloqueo UDP puerto 80 . . . . .	13
4.8. Bloqueo de puerto 5001, UDP y proveniente del host 1 . . . . .	14
4.9. Bloqueo de MAC entre los host 1 (Client) y host 3 (Server) . . . . .	15
4.9.1. TCP . . . . .	15
4.9.2. UDP . . . . .	16
4.10. Bloqueo de MAC entre los host 1 (Server) y host 3 (Client) . . . . .	17
4.10.1. TCP . . . . .	17
4.10.2. UDP . . . . .	18
4.11. Ejemplo sin pasar por firewall . . . . .	19
<b>5. Preguntas a responder</b>	<b>21</b>
5.1. ¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común? . . . . .	21
5.2. ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow? . . . . .	21
5.3. ¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow? Piense en el escenario interASes para elaborar su respuesta . . . . .	21
<b>6. Dificultades encontradas</b>	<b>22</b>
<b>7. Conclusiones</b>	<b>22</b>
<b>8. Referencias</b>	<b>22</b>

## 1. Introducción

Este trabajo tiene como objetivo abordar los desafíos que han llevado al surgimiento de las SDN y el protocolo OpenFlow, que permite programar dispositivos de red de manera programable.

El mismo se centrará en la construcción de una topología dinámica utilizando OpenFlow para implementar un Firewall a nivel de capa de enlace.

Finalmente, esta simulación se realizará utilizando Mininet y un controlador POX, lo que nos permitirá explorar y comprender en la práctica el funcionamiento de las SDN y OpenFlow.

## 2. Hipótesis/Supuestos

- Como mínimo debe haber un switch.
- El controlador POX es estable.
- La red se comportará de manera predecible bajo las reglas definidas.
- La topología de red diseñada es escalable.
- Las reglas de firewall funcionan de la manera esperada.
- Los componentes de la red (hosts, switches, controlador) funcionan sin problemas bajo OpenFlow.

### 3. Detalles de implementación

#### 3.1. Topología

El archivo `topologia.py` define una topología para una red SDN utilizando Mininet, una herramienta para simular redes. Aquí está la descripción de la topología:

- Número de Switches: La topología se configura para tener un número variable de switches ( $n$  switches). Esto permite flexibilidad para experimentar con diferentes tamaños de red.
- Hosts y Switches: Se crean 4 hosts (host1 a host4). Estos hosts se conectan a los switches. Si hay más de un switch, se conectan en serie (cada switch con su sucesor).
- Conexiones: Los hosts host1 y host2 se conectan al primer switch. Los hosts host3 y host4 se conectan al último switch. Los switches se conectan en línea, formando una cadena desde el primer hasta el último switch.
- Excepciones: Si el número de switches es menor a 1, se lanza una excepción, ya que se necesita al menos un switch para formar una red.

Este diseño de topología es bastante básico pero efectivo para entender cómo los switches SDN interactúan con múltiples hosts y entre ellos.

#### 3.2. Firewall

El archivo `firewall.py` implementa un firewall básico utilizando el controlador POX para redes SDN. El código contiene lo siguiente:

- Carga de Reglas: El firewall carga un conjunto de reglas desde un archivo JSON. Estas reglas definen qué tráfico debe ser bloqueado o permitido.
- Clase Firewall: Se define una clase Firewall que escucha eventos en la red SDN. Cuando un switch se conecta al controlador (evento `ConnectionUp`), el firewall aplica las reglas de filtrado a ese switch específico.
- Aplicación de Reglas: Cada regla se traduce en una modificación de flujo (`ofp flow mod`) en OpenFlow. Las reglas pueden especificar direcciones IP de origen y destino, puertos y direcciones MAC, así como el protocolo (TCP/UDP). Si un paquete coincide con una regla, se toma la acción correspondiente (por ejemplo, descartar el paquete).
- Función de Lanzamiento: La función `launch` inicializa el firewall con las reglas especificadas y el ID del switch al que se aplicarán las reglas.

Este script de firewall es un ejemplo práctico de cómo se pueden implementar políticas de seguridad en una red SDN utilizando el controlador POX y OpenFlow. Permite una gran flexibilidad y control sobre el tráfico de la red, lo cual es una de las ventajas clave de las redes SDN.

## 4. Pruebas

Para las pruebas, se decidió utilizar dos switches por simplicidad. Así, la topología quedó configurada de la siguiente manera:

### 4.1. Resultado de las simulaciones

- Switch 1 eth 2 (Switch1 con host1)
- Switch 1 eth 3 (Switch1 con host2)
- Switch 2 eth 2 (Switch2 con host3)
- Switch 2 eth 3 (Switch2 con host4)
- Switch 1 eth 1 (Switch1)
- Switch 2 eth 1 (Switch2)

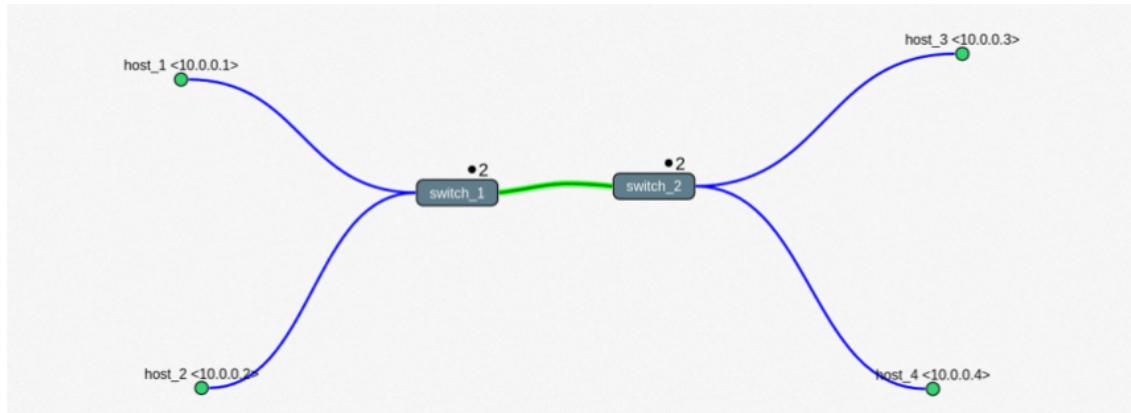


Figura 1: simulación

```
mininet> links
host_1-eth0<->switch_1-eth2 (OK OK)
host_2-eth0<->switch_1-eth3 (OK OK)
host_3-eth0<->switch_2-eth2 (OK OK)
host_4-eth0<->switch_2-eth3 (OK OK)
switch_1-eth1<->switch_2-eth1 (OK OK)
mininet> dump
<Host host_1: host_1-eth0:10.0.0.1 pid=40210>
<Host host_2: host_2-eth0:10.0.0.2 pid=40212>
<Host host_3: host_3-eth0:10.0.0.3 pid=40214>
<Host host_4: host_4-eth0:10.0.0.4 pid=40216>
<OVSSwitch switch_1: lo:127.0.0.1,switch_1-eth1:None,switch_1-eth2:None,switch_1-eth3:None pid=40221>
<OVSSwitch switch_2: lo:127.0.0.1,switch_2-eth1:None,switch_2-eth2:None,switch_2-eth3:None pid=40224>
<RemoteController c0: 127.0.0.1:6633 pid=40202>
```

Figura 2: resultado de mininet>links y mininet>dump

Para todas las pruebas, se puede visualizar todos los comandos utilizados. Cabe destacar que para desplegar las terminales de un host dentro de mininet se utilizó el comando xterm. Una vez en éstas terminales, aplicamos los comandos iperf con los parámetros correspondientes como muestra cada una de las imágenes.

## 4.2. Prueba Pingall

```

nahuel@nahuel-desktop:~/Documents/sistemasDistribuidos/TP2-SDN$ sudo mn --custom topologia.py --topo topo
logia --switch ovsk --controller remote
Inicializando la topología con 2 switches
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** Adding hosts:
host_1 host_2 host_3 host_4
*** Adding switches:
switch_1 switch_2
*** Adding links:
(host_1, switch_1) (host_2, switch_1) (host_3, switch_2) (host_4, switch_2) (switch_1, switch_2)
*** Configuring hosts
host_1 host_2 host_3 host_4
*** Starting controller
c0
*** Starting 2 switches
switch_1 switch_2 ...
*** Starting CLI:
mininet> links
host_1-eth0<->switch_1-eth2 (OK OK)
host_2-eth0<->switch_1-eth3 (OK OK)
host_3-eth0<->switch_2-eth2 (OK OK)
host_4-eth0<->switch_2-eth3 (OK OK)
switch_1-eth1<->switch_2-eth1 (OK OK)
mininet> dump
<Host host_1: host_1-eth0:10.0.0.1 pid=40210>
<Host host_2: host_2-eth0:10.0.0.2 pid=40212>
<Host host_3: host_3-eth0:10.0.0.3 pid=40214>
<Host host_4: host_4-eth0:10.0.0.4 pid=40216>
<OVSSwitch switch_1: lo:127.0.0.1,switch_1-eth1:None,switch_1-eth2:None,switch_1-eth3:None pid=40221>
<OVSSwitch switch_2: lo:127.0.0.1,switch_2-eth1:None,switch_2-eth2:None,switch_2-eth3:None pid=40224>
<RemoteController c0: 127.0.0.1:6633 pid=40202>
mininet> pingall
*** Ping: testing ping reachability
host_1 -> host_2 host_3 host_4
host_2 -> host_1 host_3 host_4
host_3 -> host_1 host_2 host_4
host_4 -> host_1 host_2 host_3
*** Results: 0% dropped (12/12 received)
mininet> 

```

Figura 3: resultado de mininet &gt;pingall

Como no había ningún firewall activado, no se pierde ningún paquete.

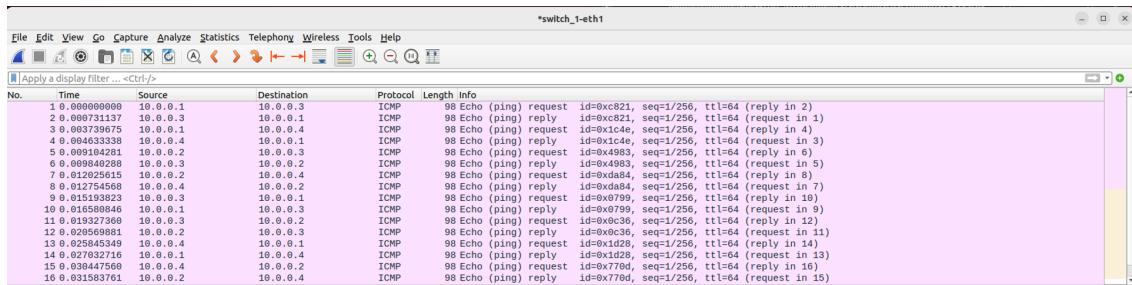


Figura 4: Captura wireshark sobre switch\_1-eth1

La captura de Wireshark en la interfaz switch\_1-eth1 nos permite observar los paquetes en-

viados del host 1 y 2 al 3 y 4, y viceversa. Los paquetes que van del host 1 al 2 y viceversa, y del 3 al 4 y viceversa, no pasan por esta interfaz, y por lo tanto no son capturados.

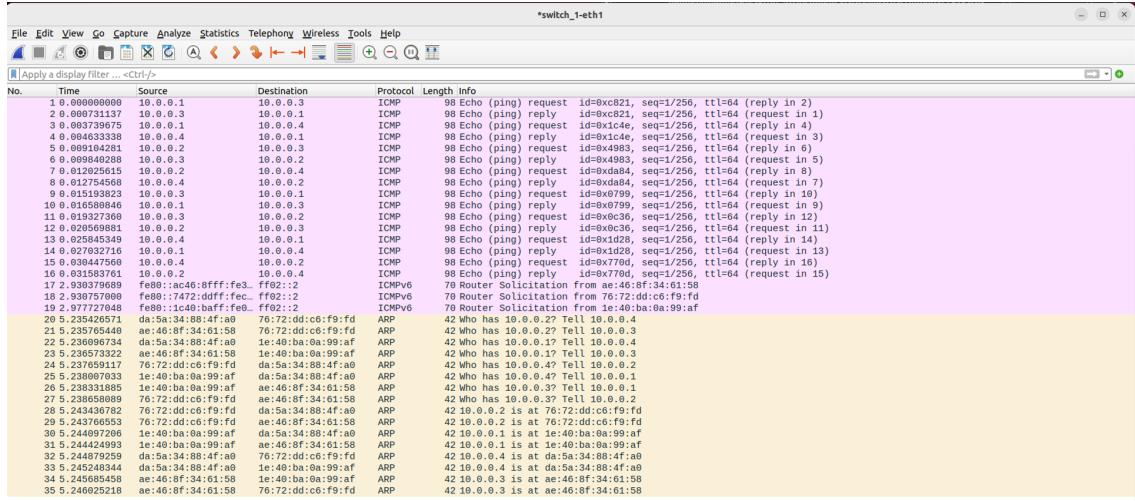


Figura 5: Captura wireshark con más información

```
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-02 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-02 2]
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 3]
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.2 -> 00:00:00:00:00:02.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.3 -> 00:00:00:00:00:01.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.2 -> 00:00:00:00:00:03.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:03.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.1 -> 00:00:00:00:00:01.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.2 -> 00:00:00:00:00:04.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:04.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.3 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.1 -> 00:00:00:00:00:01.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.3 -> 00:00:00:00:00:01.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.2 -> 00:00:00:00:00:02.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.3 -> 00:00:00:00:00:03.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.1 -> 00:00:00:00:00:03.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.2 -> 00:00:00:00:00:02.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.1 -> 00:00:00:00:00:02.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.3 -> 00:00:00:00:00:04.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.1 -> 00:00:00:00:00:04.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.3 -> 00:00:00:00:00:02.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.1 -> 00:00:00:00:00:02.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.2 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.1 -> 00:00:00:00:00:01.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.2 -> 00:00:00:00:00:03.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:03.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.2 -> 00:00:00:00:00:02.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.1 -> 00:00:00:00:00:02.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.3 -> 00:00:00:00:00:03.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.1 -> 00:00:00:00:00:03.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:03.2 -> 00:00:00:00:00:04.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.3 -> 00:00:00:00:00:03.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.3 -> 00:00:00:00:00:01.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.1 -> 00:00:00:00:00:01.2
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.2 -> 00:00:00:00:00:04.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:01.1 -> 00:00:00:00:00:04.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.3 -> 00:00:00:00:00:02.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.1 -> 00:00:00:00:00:02.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.3 -> 00:00:00:00:00:04.1
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:02.1 -> 00:00:00:00:00:04.3
DEBUG:forwarding.l2_learning:installing flow for 00:00:00:00:00:04.3 -> 00:00:00:00:00:03.2
```

Figura 6: Registros del controlador remoto

### 4.3. Firewall - Aplicación de reglas

```
mahui@nahui:~/Desktop:/Documents/sistemasDistribuidos/TP2-SDN$ sudo python3 pox.py log.level --DEBUG log.color openflow.of_01 forwarding.l2_learning firewall --path_reglas="reglas.json"
POX 0.7.9 (gar) / Copyright 2011-2020 James McCauley, et al.
DEBUG:firewall:Reglas cargadas: ['reglas': [{'protocol': 'tcp', 'dst_port': 80}, {'protocol': 'udp', 'dst_port': 80}, {'protocol': 'udp', 'dst_port': 5001, 'src_ip': '10.0.0.1'}, {'src_mac': '00:00:00:00:00:03', 'dst_mac': '00:00:00:00:00:02'}]}
DEBUG:firewall:Habilitando el Firewall
DEBUG:core:POX 0.7.9 (gar) going up...
DEBUG:core:Version: Python 3.6.2+ Jun 11 2023 05:26:28
DEBUG:core:Platform: Linux-6.2.0-36-generic-x86_64-with-glibc2.35
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:listening on 0.0.0.0:6633
INFO:openflow.of_01:Switch 1 connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-02 2] connected
INFO:openflow.of_01:[00-00-00-00-00-01 3] connected
DEBUG:forwarding.l2_learning:connection [00-00-00-00-00-01 3]
DEBUG:firewall:Regla instalada: droppeando paquete TCP
DEBUG:firewall:Regla instalada: droppeando paquete con puerto de destino 80
DEBUG:firewall:Regla instalada: droppeando paquete UDP
DEBUG:firewall:Regla instalada: droppeando paquete con puerto de destino 80
DEBUG:firewall:Regla instalada: droppeando paquete UDP
DEBUG:firewall:Regla instalada: droppeando paquete con puerto de destino 5001
DEBUG:firewall:Regla instalada: droppeando paquete proveniente de dirección IP 10.0.0.1
DEBUG:firewall:Regla instalada: droppeando paquete proveniente de dirección MAC 00:00:00:00:00:03
DEBUG:firewall:Regla instalada: droppeando paquete con dirección MAC destino 00:00:00:00:00:03
DEBUG:firewall:Regla instalada: droppeando paquete proveniente de dirección MAC 00:00:00:00:00:03
DEBUG:firewall:Regla instalada: droppeando paquete con dirección MAC destino 00:00:00:00:00:02
DEBUG:firewall:Reglas de Firewall instaladas en 00-00-00-00-00-01 - switch 1
```

Figura 7: muestra de las reglas por consola

Las reglas establecidas son:

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 debe elegir dos hosts cualquiera, y los mismos no deben poder comunicarse de ninguna forma.

En primer lugar, para descartar los mensajes cuyo puerto destino sea 80, fue necesario aplicar 2 reglas: una indicando el protocol: TCP y dst \_port: 80, y otra protocol: UDP y dst \_port: 80. Fue necesario indicar ambos protocolos ya que sino el controlador remoto lanza un error de precaución donde existen parámetros no especificados.

En segundo lugar, para descartar los mensajes con estas reglas se establece: src\_ip: 10.0.0.1 para indicar que provienen del host 1, dst\_port: 5001 para indicar que el puerto destino es 5001 y protocol:UDP para indicar que el protocolo utilizado es UDP.

En tercer lugar, se utilizaron los hosts 1 y 3, y a partir de su dirección MAC, se imposibilita la conexión entre ellos de ninguna forma. Cabe destacar que se indica src\_mac y dst\_mac para ambos hosts para imposibilitar tanto la comunicación de un lado al otro y viceversa.

Aplicaremos estas reglas en el switch 1, establecido para ser el firewall. Cabe mencionar que se podría elegir cualquier otro simplemente pasando por parámetro el id de switch al que se le quiere aplicar las reglas para que actúe como firewall. Para comprobar el correcto funcionamiento del firewall se hizo uso de iperf sobre mininet para enviar paquetes sobre la red.

#### 4.4. l2 learning

```

DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.2 -> 9a:4e:59:10:13:f5.1
DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.2 -> e6:4c:0a:83:e7:3c.3
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.1 -> e6:4c:0a:83:e7:3c.2
DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.2 -> 9a:4e:59:10:13:f5.1
DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.1 -> 9a:4e:59:10:13:f5.3
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.2
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.2
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.2
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.1 -> e6:4c:0a:83:e7:3c.2
DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.2 -> 9a:4e:59:10:13:f5.1
DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.2 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.3 -> e6:4c:0a:83:e7:3c.1
DEBUG:forwarding.l2_learning:installing flow for 9a:4e:59:10:13:f5.1 -> e6:4c:0a:83:e7:3c.2
DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.2 -> 9a:4e:59:10:13:f5.1
DEBUG:forwarding.l2_learning:installing flow for e6:4c:0a:83:e7:3c.2 -> e6:4c:0a:83:e7:3c.1

```

Figura 8: muestra de l2 learning de pox por consola

## 4.5. Conexión exitosa

Probaremos levantando un servidor tanto TCP como UDP en el host 2 y vamos a conectarnos a él desde el host 4. Esta interacción no está prohibida por el firewall que se configuró por lo que el servidor debería recibir los paquetes correctamente.

### 4.5.1. UDP

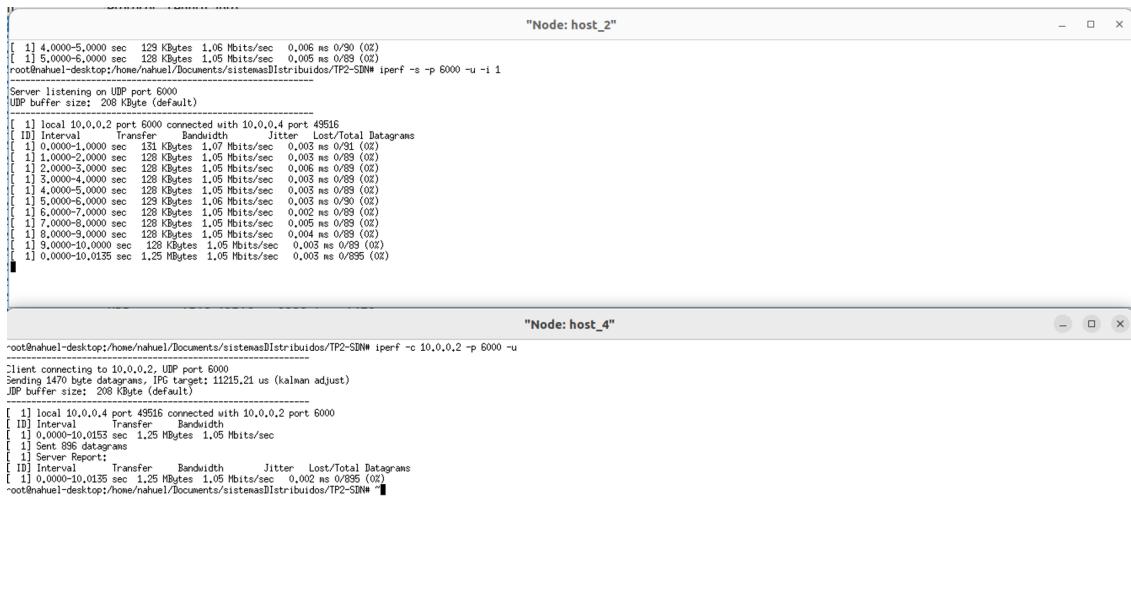


Figura 9: conexión exitosa usando UDP

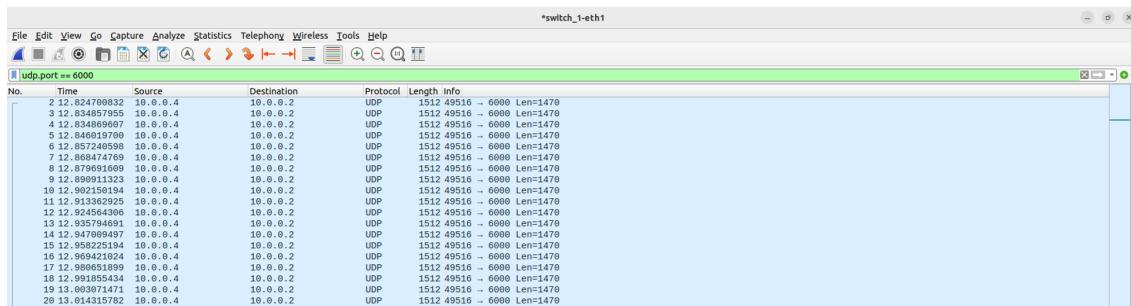


Figura 10: conexión exitosa usando UDP por Wireshark sobre interfaz switch\_1-eth1

#### 4.5.2. TCP

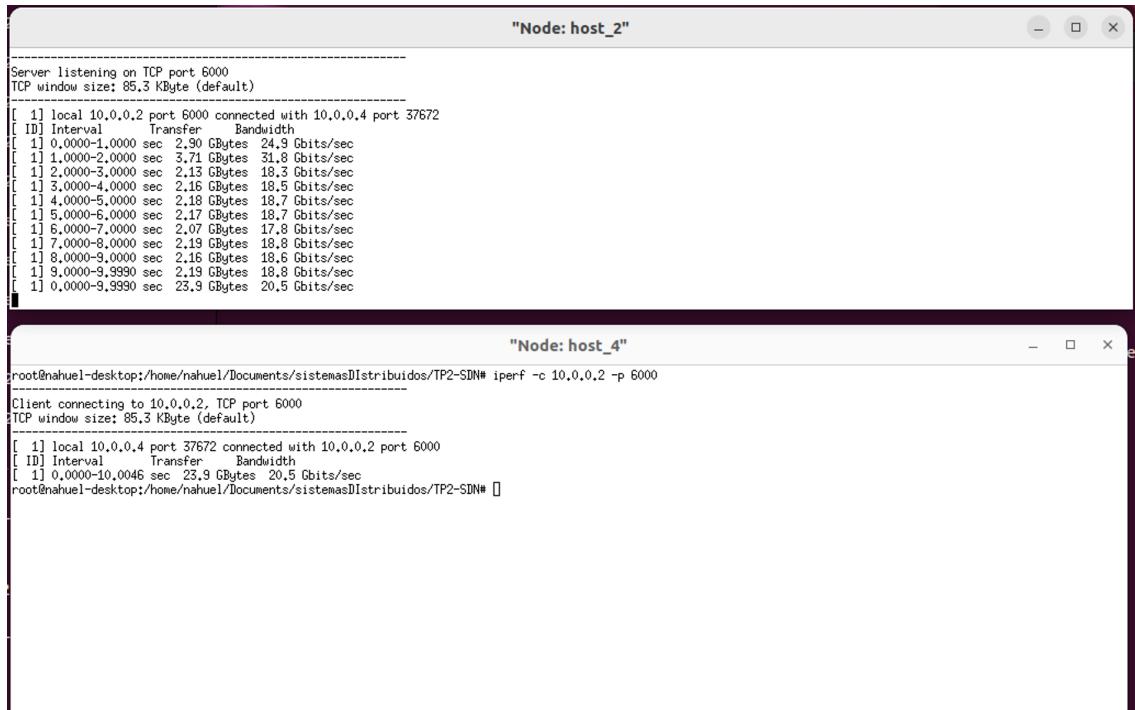


Figura 11: conexión exitosa usando TCP

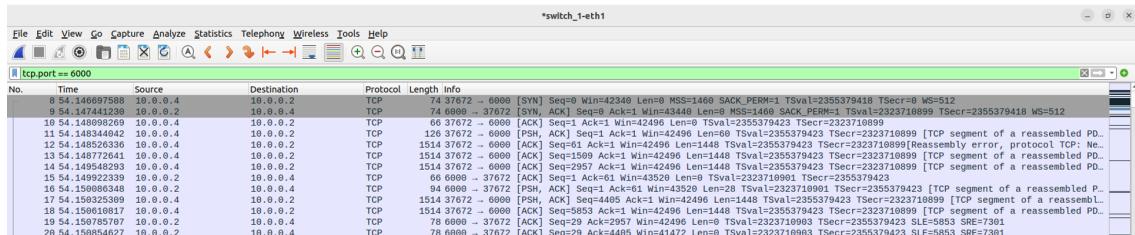


Figura 12: conexión exitosa usando TCP por Wireshark sobre interfaz switch\_1-eth1

Se mostrará aquí una simulación entre el host 2 como servidor y el host 4 como cliente con los casos en los cuales el Firewall debe bloquear el envío de paquetes destinados al puerto 80.

#### 4.6. Bloqueo TCP puerto 80

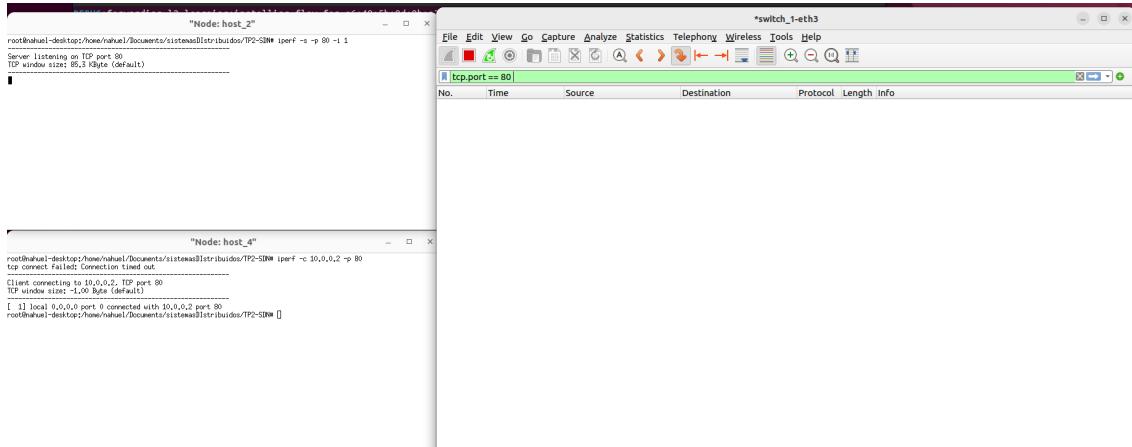


Figura 13: Al switch1-eth3 no pasan datos

Como se puede ver, en este caso el servidor 2 no recibe nada. Esto se debe a que el firewall entró en acción bloqueando los paquetes del host 4 ya que cumplen con la primera regla del firewall, y por lo tanto, el host 4 retransmite paquetes hasta llegar el timeout. Si vemos la interfaz del switch 1 que directamente está linkada con el host 2 vemos que efectivamente no recibe ningún paquete que tenga como destino el puerto 80 y el protocolo sea TCP.

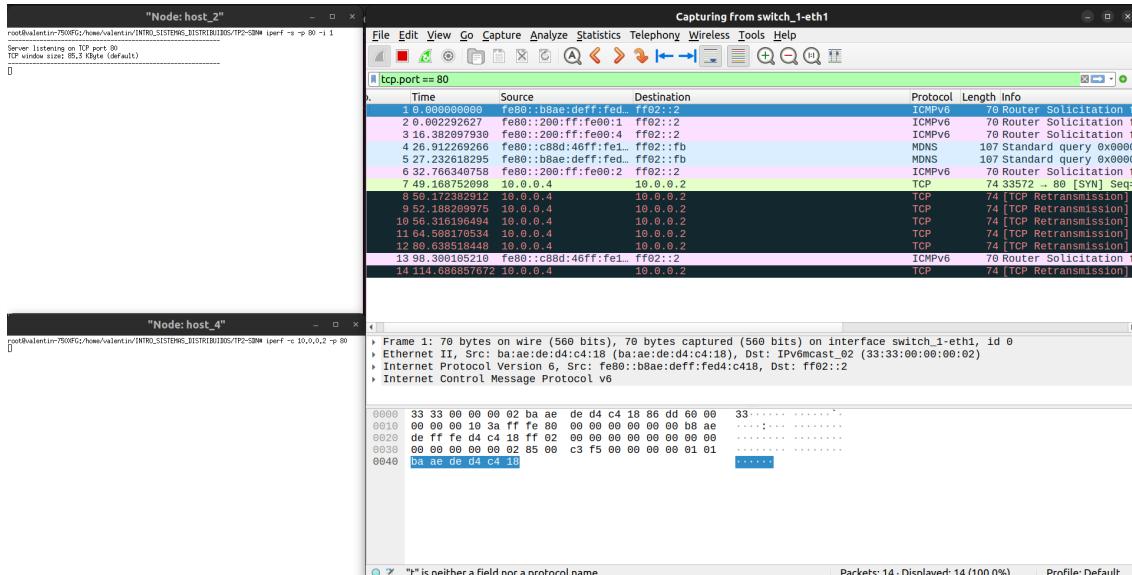


Figura 14: Al switch1-eth1 le llegan los mensajes de retransmisión de TCP

## 4.7. Bloqueo UDP puerto 80

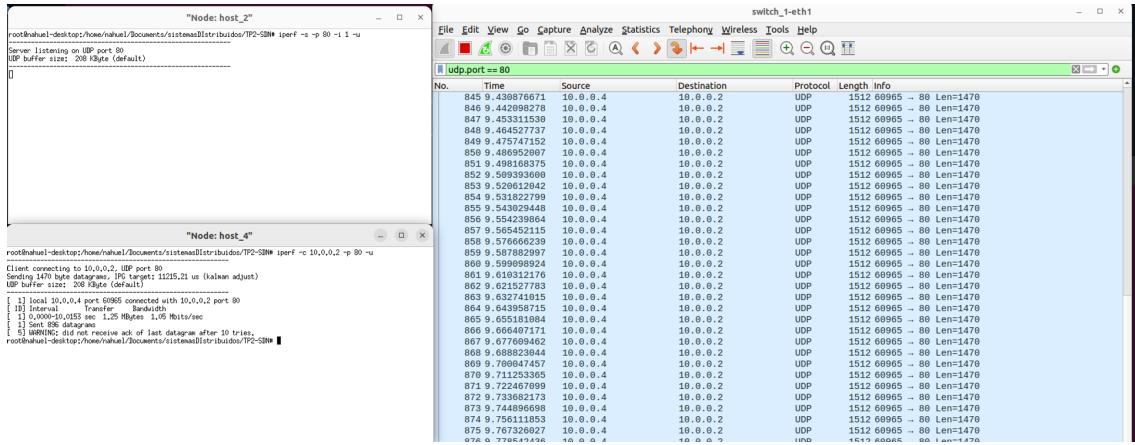


Figura 15: Al switch1-eth1 llegan los mensajes UDP y se droppean.

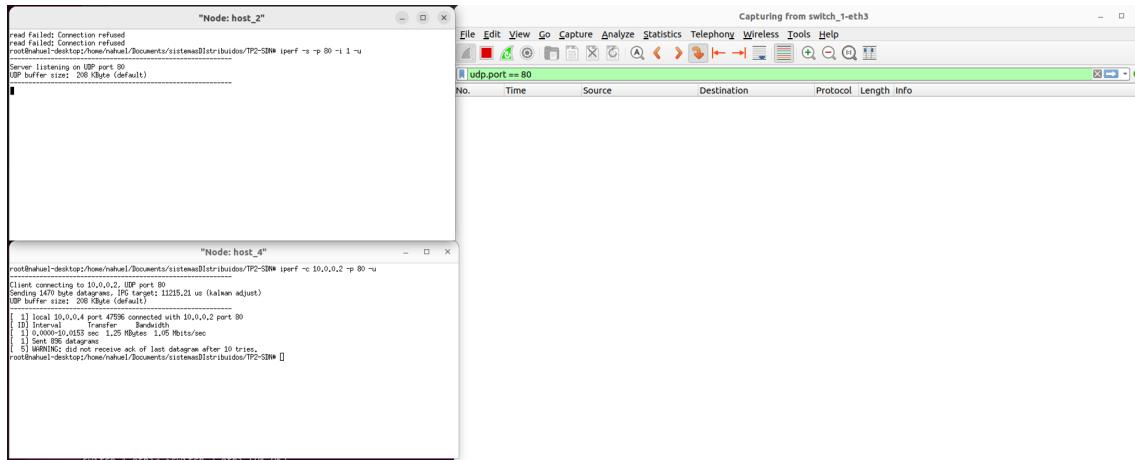


Figura 16: Al switch1-eth3 no pasan mensajes

#### 4.8. Bloqueo de puerto 5001, UDP y proveniente del host 1

Ahora, haremos una prueba con un servidor UDP en host 2 en el puerto 5001 con el host 1 como cliente. A priori, esta comunicación no debería funcionar ya que se cumple que el puerto de destino sea el 5001, el protocolo sea UDP y el host cliente es el 1.

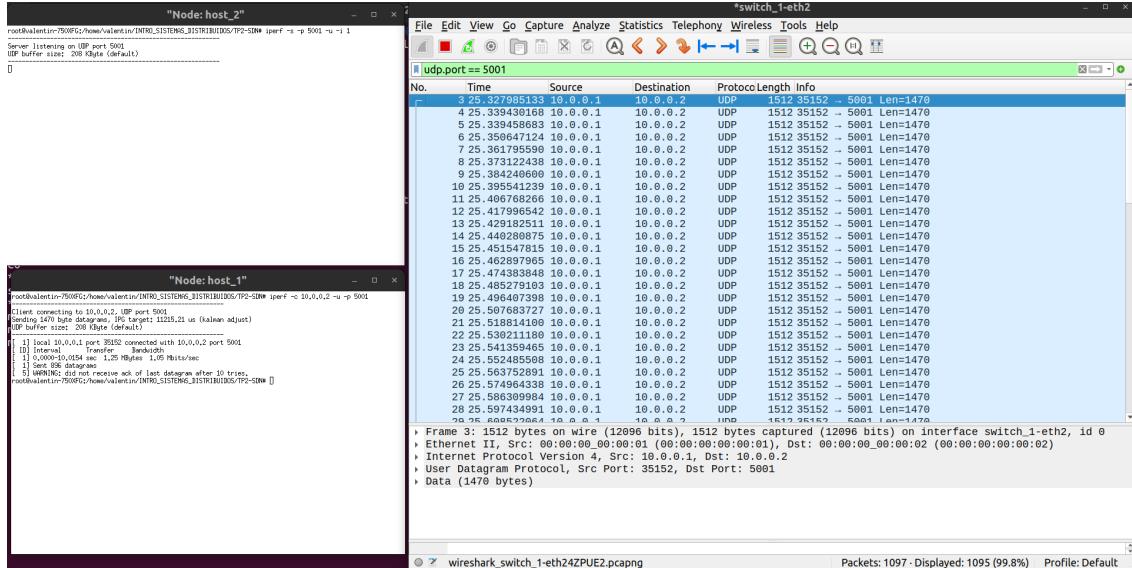


Figura 17: Prueba con switch1-eth2 - Paquetes llegan al switch 1

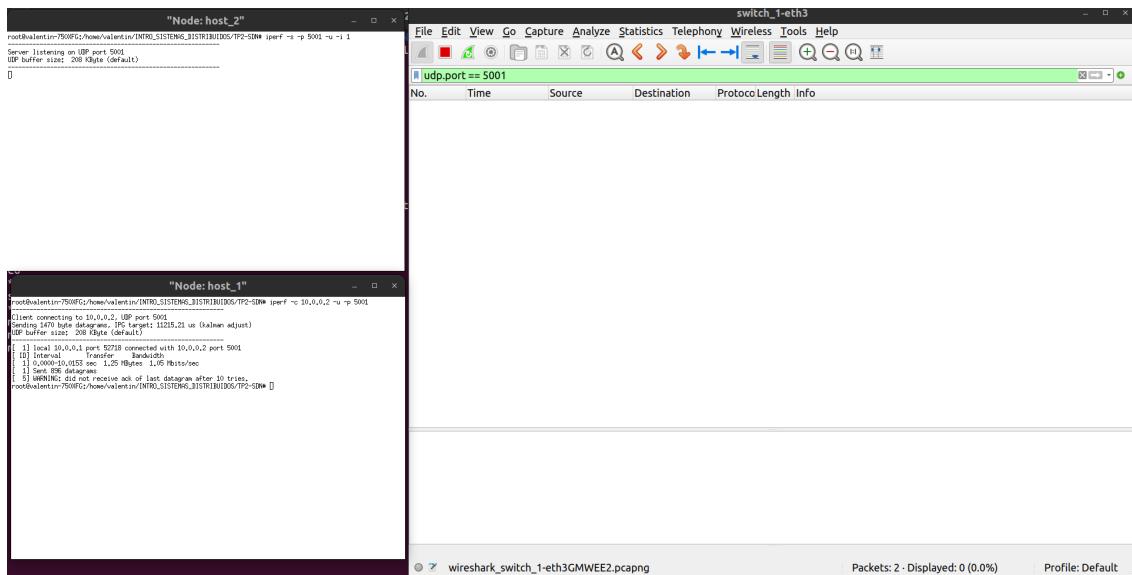


Figura 18: Vista desde switch1-eth3 - No llega ning n paquete

En este caso vamos a probar enviar paquetes entre el host 1 y el host 3. Como se estableció en las reglas, estos hosts no deberían poder comunicarse de ninguna forma. Para ello, se probará tanto al host 1 como cliente y host 3 como servidor y viceversa.

## 4.9. Bloqueo de MAC entre los host 1 (Client) y host 3 (Server)

### 4.9.1. TCP

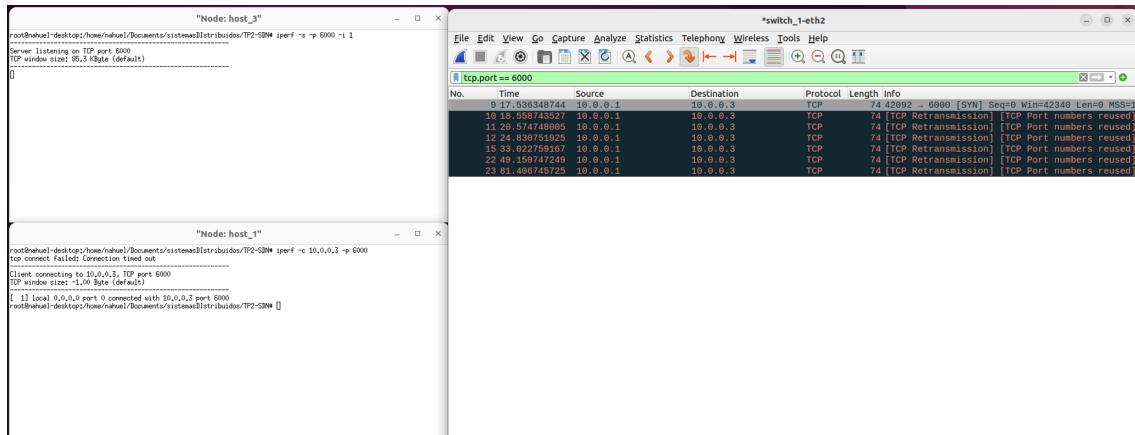


Figura 19: Llegan los mensajes al switch1-eth2 para luego ser filtrados

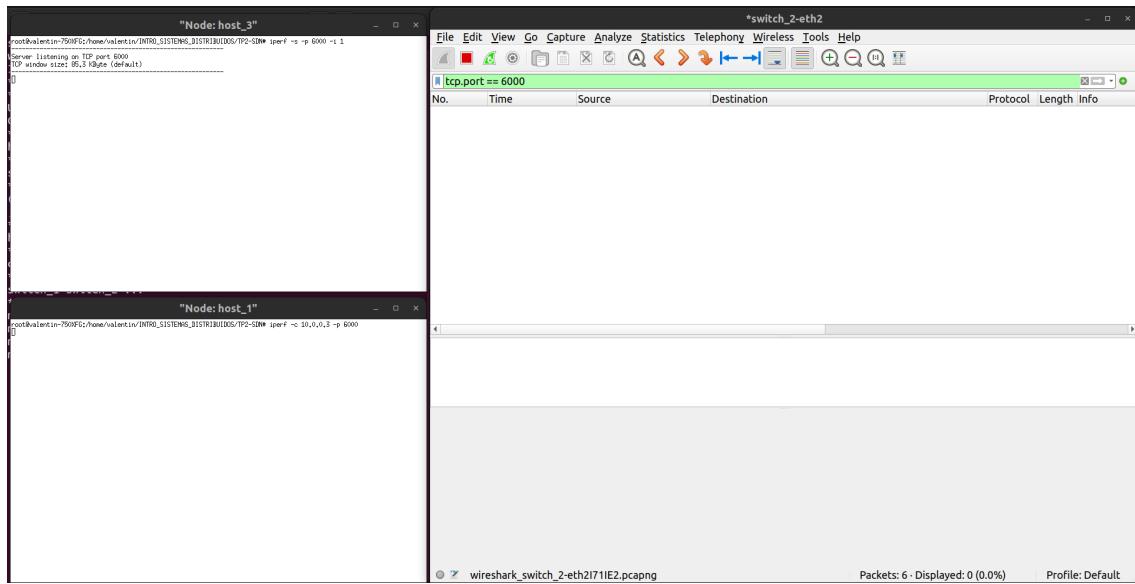


Figura 20: No llegan los mensajes al switch2-eth2

#### 4.9.2. UDP

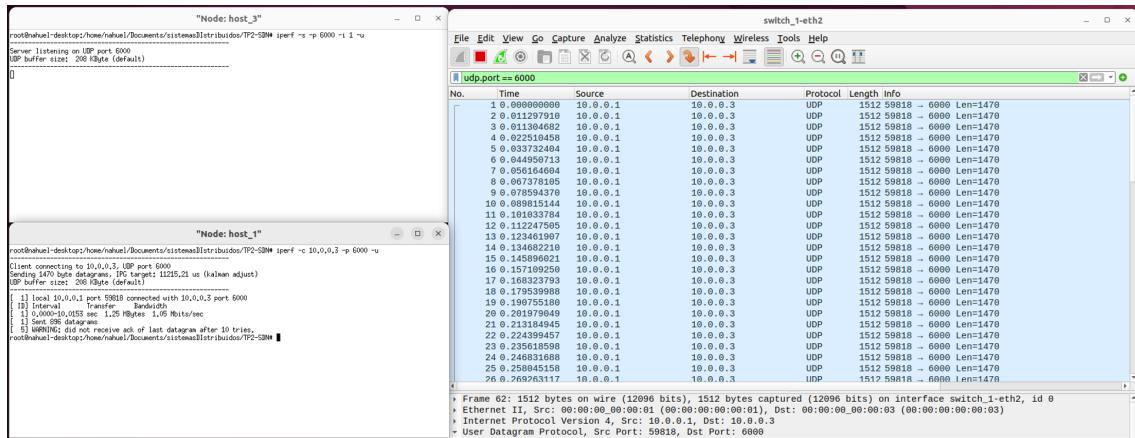


Figura 21: Llegan al switch1-eth2 y los droppea

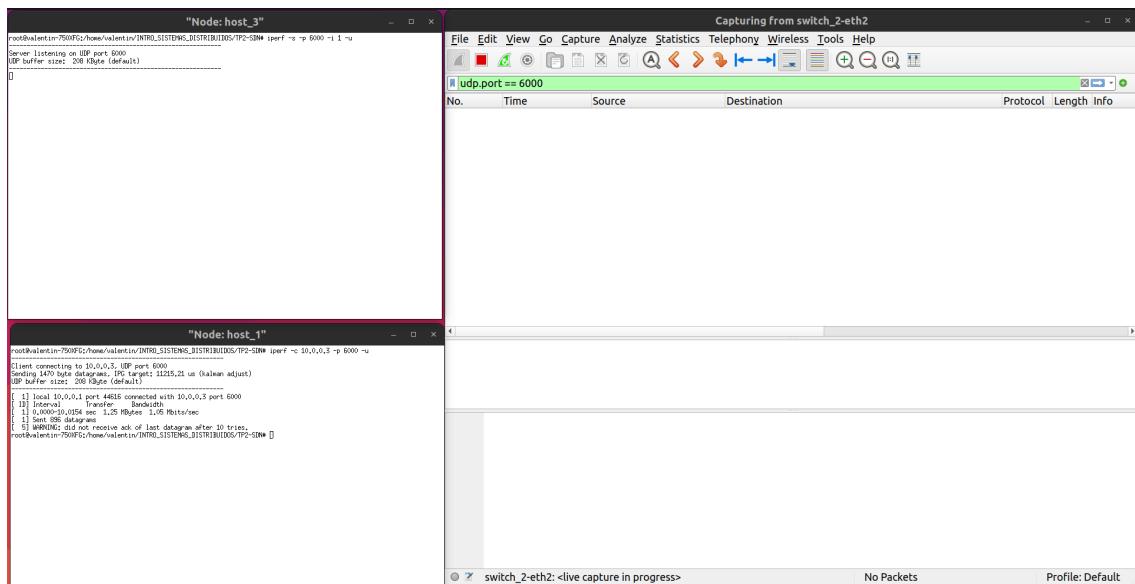


Figura 22: No llegan paquetes al switch2-eth2

## 4.10. Bloqueo de MAC entre los host 1 (Server) y host 3 (Client)

### 4.10.1. TCP

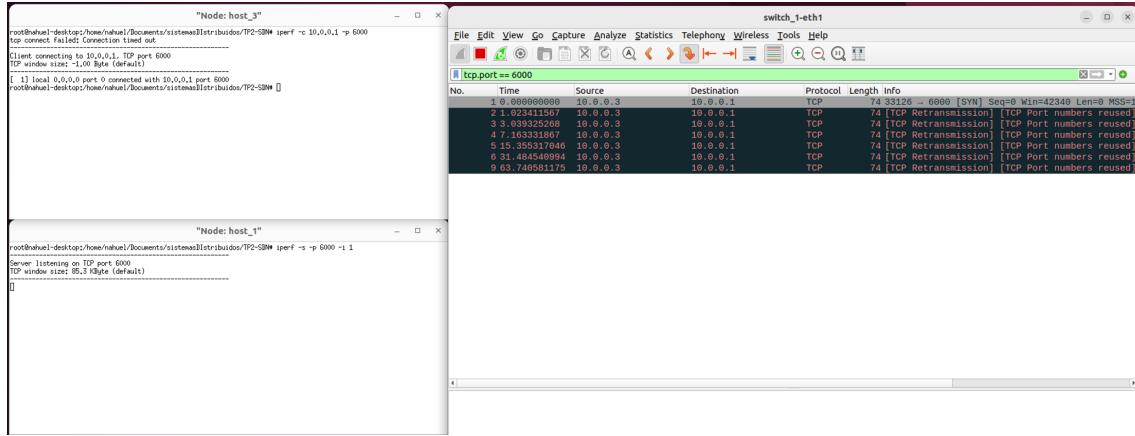


Figura 23: Llegan al switch1-eth1 se droppean por el firewall

Vemos que ningún paquete llega al host 1 y que llegan los paquetes de retransmisión al switch 1 por medio de la interfaz switch\_1-eth1 por lo cual el firewall está cumpliendo su función.

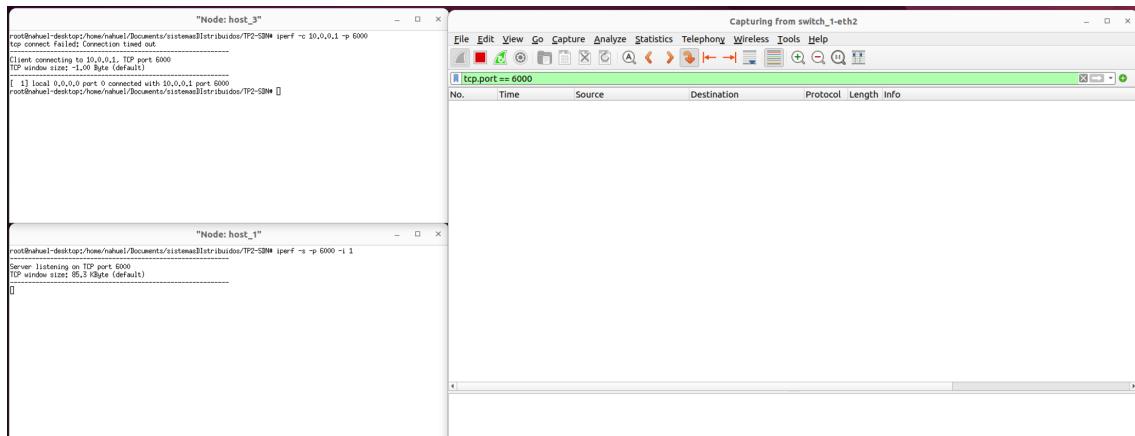


Figura 24: No llegan mensajes al switch1-eth2 (conecta al switch con firewall y el host 1)

#### 4.10.2. UDP

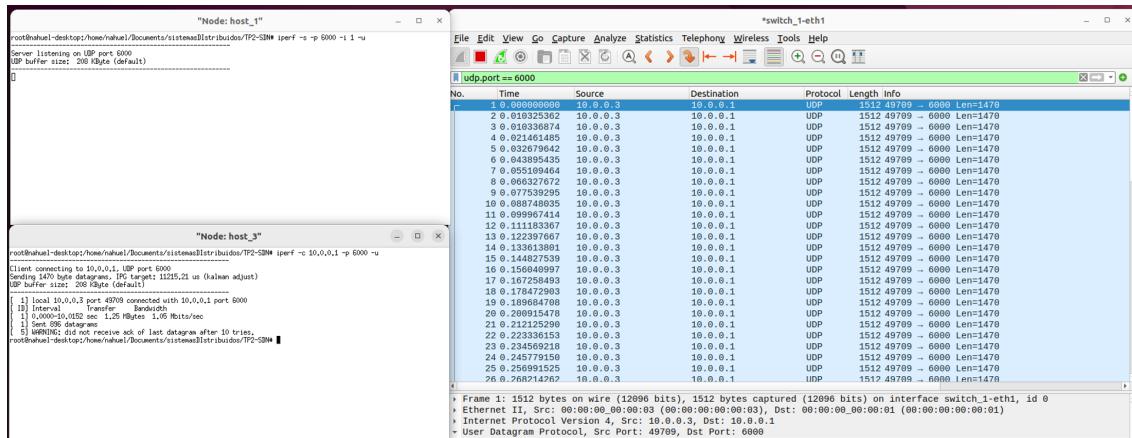


Figura 25: Llegan paquetes de retransmisión al switch1-eth1. Paquetes al host 1 se droppean por el firewall

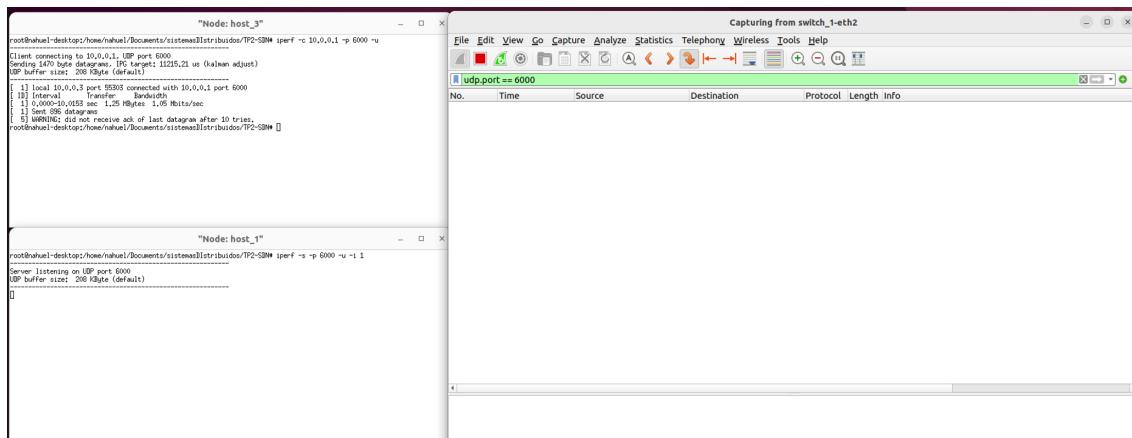


Figura 26: No llegan mensajes al switch1-eth2 (conecta al switch con firewall y el host 1)

#### 4.11. Ejemplo sin pasar por firewall

Probaremos ahora un ejemplo donde el tráfico no pase por el firewall. Entonces, si bien el firewall, de acuerdo a las reglas definidas, debe bloquear todo el tráfico que esté destinado al puerto 80, esto no significa que se descarte el tráfico si el mismo no pasa por el switch configurado con el firewall. En particular, si intentamos comunicar mediante TCP al host 3 como servidor y al host 4 como cliente entonces deberíamos observar una comunicación exitosa.

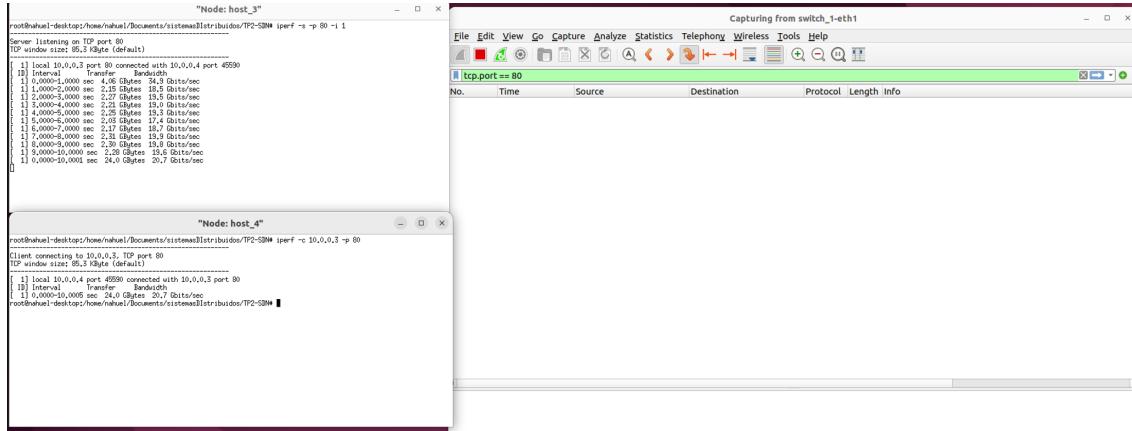


Figura 27: No llegan paquetes al switch con firewall

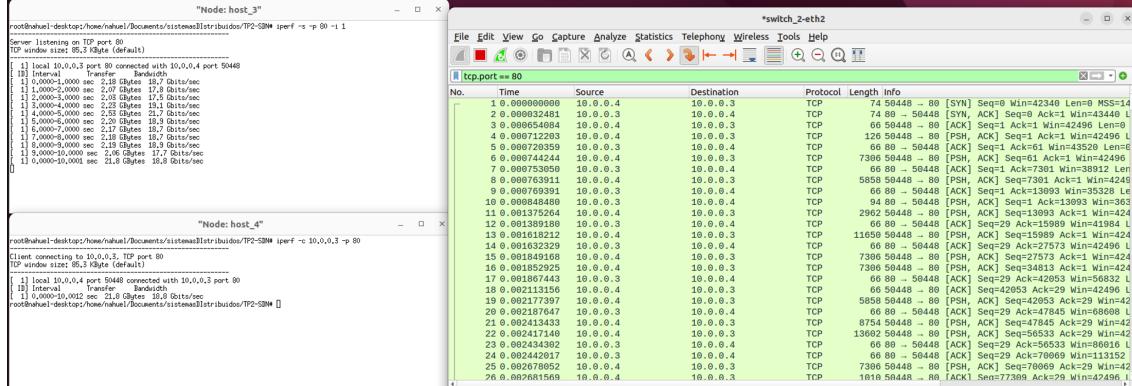


Figura 28: Paquetes al switch2-eth2

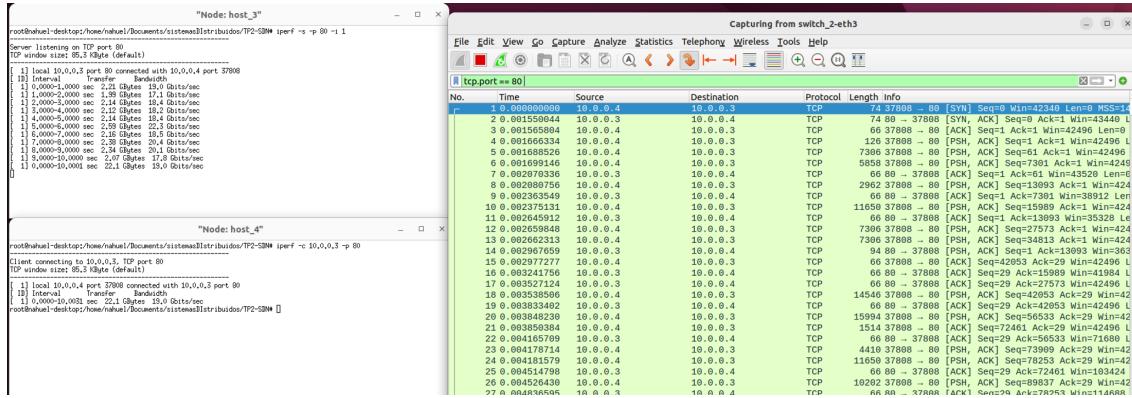


Figura 29: Paquetes al switch2-eth3

## 5. Preguntas a responder

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

- Un switch opera principalmente en la capa de enlace de datos (capa 2) del modelo TCP/IP. Su función principal es el reenvío de paquetes en una red local (LAN) basándose en las direcciones MAC de los dispositivos conectados en esa LAN. Los switches son utilizados para segmentar y mejorar la eficiencia de las redes locales.
- Un router opera en la capa de red (capa 3) del modelo TCP/IP. Su función principal es el enrutamiento de paquetes entre diferentes redes, como la conexión entre una LAN y la Internet. Los routers toman decisiones de enrutamiento basadas en direcciones IP y permiten que los paquetes viajen entre redes con diferentes direcciones IP.

Por otro lado, los switches y los routers tienen en común que permiten conectar diferentes dispositivos entre sí. Ambos dispositivos son capaces de comutar paquetes de datos y pueden determinar por dónde tiene que salir un paquete en función de sus direcciones de origen y destino.

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

- Un switch convencional en el modelo TCP/IP opera de manera tradicional, tomando decisiones de reenvío de paquetes basadas en la dirección MAC de destino en la capa de enlace de datos. No tiene la capacidad de tomar decisiones de reenvío más allá de estas direcciones MAC. Las funcionalidades asociadas al plano de control y el de datos se realizan en el mismo dispositivo.
- Un switch OpenFlow es parte de un enfoque de red definida por software (SDN) que permite la programación centralizada y dinámica del comportamiento de los dispositivos de red. En lugar de tomar decisiones de reenvío en función de la dirección MAC, un switch OpenFlow utiliza el protocolo OpenFlow para recibir instrucciones de un controlador SDN centralizado. Esto permite una mayor flexibilidad y control en la gestión de la red.

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

Pensando en el escenario interASes, los routers que permiten la comunicación deben implementar el protocolo BGP. Entonces, se necesitaría que cada dispositivo posea una gran cantidad de entradas BGP almacenadas. Los switches OpenFlow son valiosos en entornos de redes locales y en ciertos casos de SDN, pero no pueden reemplazar la funcionalidad completa de los routers en la Internet global. Los routers son cruciales para el enrutamiento de paquetes a nivel mundial y la interconexión de redes autónomas. Por lo tanto, ambos dispositivos, routers y switches OpenFlow, tienen roles diferentes y son complementarios en la infraestructura de Internet.

## 6. Dificultades encontradas

- Comprensión de Conceptos SDN y OpenFlow, es decir, cómo los controladores SDN interactúan con los switches y cómo se manejan los flujos de datos en la red.
- Desarrollo del Firewall con POX
- Integración de la topología de la red con el controlador y el firewall

## 7. Conclusiones

Este proyecto nos ha fortalecido la comprensión técnica de las redes definidas por software y sobre el protocolo OpenFlow. Lo más destacado fue en como los conceptos teóricos se veían en la implementación práctica, es decir, la capacidad de controlar y manipular el tráfico de red gracias a la correcta configuración de la topología y las efectivas reglas del firewall, ha ampliado nuestro horizonte sobre el concepto de las redes.

## 8. Referencias

- [Mininet](#)
- [POX](#)
- [OpenFlow](#)
- [Open vSwitch](#)
- [Visualizador de topologías](#)