

REDES: INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS
(TA048) CURSO: ALVAREZ HAMELIN

Trabajo Práctico 2 Software-Defined Networks

× ×

Junio 2025

Agustín Altamirano	110237
Cristhian David Noriega	109164
Ian von der Heyde	107638
Juan Martín de la Cruz	109588
Santiago Tomás Fassio	109463

Índice

1. Introducción	3
2. Implementación y topología	4
2.1. Topología	4
2.2. Firewall	4
2.2.1. Reglas de filtrado aplicadas en el firewall	5
2.2.2. Evidencia de instalación de reglas en el switch central	6
3. Pruebas	7
3.1. Prueba 0: Conectividad básica sin firewall	8
3.2. Prueba 1: Bloqueo de comunicación entre h1 y h3	9
3.2.1. Prueba TCP: h1 como cliente, h3 como servidor	9
3.2.2. Prueba TCP: h3 como cliente, h1 como servidor	10
3.2.3. Prueba UDP: h1 como cliente, h3 como servidor	11
3.2.4. Prueba UDP: h3 como cliente, h1 como servidor	12
3.2.5. Análisis de resultados	13
3.3. Prueba 2: Bloqueo de tráfico UDP en puerto 5001 desde h1	14
3.3.1. Tráfico UDP bloqueado: h1 hacia h4 en puerto 5001	14
3.3.2. Tráfico UDP permitido: h2 hacia h4 en puerto 5001	15
3.3.3. Análisis de resultados	17
3.4. Prueba 4: Bloqueo de tráfico HTTP (puerto 80) TCP y UDP	17
3.4.1. Bloqueo de tráfico TCP en puerto 80	17
3.4.2. Bloqueo de tráfico UDP en puerto 80	18
3.4.3. Análisis de resultados	19
3.5. Análisis general de resultados	19
4. Preguntas	20
4.1. ¿Cual es la diferencia entre un Switch y un router? ¿Que tienen en comun?	20
4.2. ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?	20
4.3. ¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow? Piense en el escenario interASes para elaborar su respuesta	21
5. Conclusiones	21
5.1. Aprendizajes sobre SDN y OpenFlow	21
5.2. Experiencia con la API de POX	22
5.3. Desafíos del desarrollo del firewall	22
5.4. Herramientas de análisis y validación	22
5.5. Aplicabilidad en entornos reales	22

1. Introducción

En las últimas décadas, Internet ha experimentado un crecimiento exponencial tanto en su infraestructura como en sus demandas. La aparición masiva de servicios multimedia, la migración hacia entornos cloud, y la incorporación de miles de millones de dispositivos conectados —principalmente gracias a los smartphones—, han desafiado los paradigmas tradicionales de diseño y administración de redes.

Frente a este escenario, surgen las **Software-Defined Networks (SDN)** como una solución innovadora que propone una separación clara entre el plano de control y el plano de datos de la red. En este nuevo modelo, los dispositivos de red como switches y routers dejan de tomar decisiones de forma autónoma y son controlados de manera centralizada por un *controlador*, lo que permite una administración más flexible, dinámica y programable.

Uno de los pilares fundamentales para habilitar esta arquitectura es el protocolo **OpenFlow**, que permite a los controladores interactuar directamente con las tablas de flujo de los switches, definiendo políticas de enrutamiento, seguridad, calidad de servicio y más, en función de múltiples características de los paquetes.

El presente trabajo práctico tiene como objetivo explorar y aplicar estos conceptos en un entorno simulado, utilizando herramientas modernas orientadas a redes definidas por software. Para ello, se utilizará:

- **Mininet**: un simulador ligero de redes que permite definir topologías complejas de manera programática y ejecutar entornos de prueba realistas para tecnologías SDN.
- **POX**: un controlador OpenFlow escrito en Python que permite desarrollar y ejecutar aplicaciones SDN personalizadas, como switches inteligentes o firewalls programables.
- **Wireshark**: una herramienta gráfica para la captura y análisis de tráfico en tiempo real, útil para inspeccionar el comportamiento de la red y validar reglas definidas.
- **ping**: una utilidad clásica para verificar conectividad entre nodos, basada en el envío de paquetes ICMP.
- **iperf**: una herramienta para generar tráfico UDP o TCP y medir el rendimiento de la red, clave para probar políticas de firewall o control de flujo.

En este informe se describen las etapas de diseño, implementación y validación de una topología definida por software, junto con la construcción de un controlador POX que actúa como firewall a nivel de capa 2 y 3, aplicando políticas de seguridad sobre flujos de red específicos. Se presentan además evidencias experimentales del funcionamiento del sistema, como logs del controlador, capturas de tráfico y resultados de pruebas de conectividad y rendimiento.

2. Implementación y topología

2.1. Topología

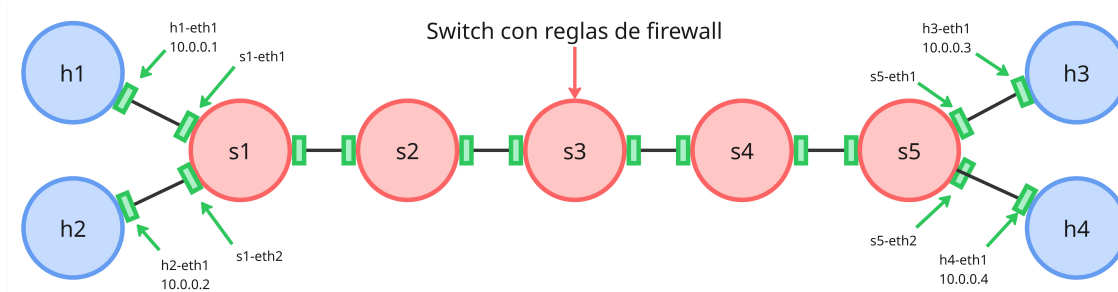


Figura 1: Topología

La topología implementada consiste en una red lineal formada por una cantidad parametrizable de switches conectados en serie (en el ejemplo de la imagen se generó con cinco switches). En los extremos de la topología se ubican los hosts: los hosts **h1** y **h2** se conectan al primer switch, mientras que los hosts **h3** y **h4** se conectan al último switch. Los switches intermedios únicamente se enlazan entre sí, formando una cadena continua.

Esta estructura fue definida mediante un script en Python utilizando la librería Mininet, lo que permite crear y modificar la cantidad de switches de manera sencilla. La parametrización facilita la experimentación con diferentes tamaños de red, permitiendo analizar el comportamiento del tráfico a medida que aumenta la cantidad de saltos entre los extremos.

Durante las pruebas, se emplearon herramientas como **ping** e **iperf** para verificar la conectividad y medir el rendimiento entre los distintos hosts. El uso de Mininet permitió levantar terminales para cada host y analizar el comportamiento de la red bajo diversos escenarios de tráfico y configuraciones. Los detalles y resultados de estas pruebas se desarrollarán en secciones posteriores del informe.

2.2. Firewall

Para la implementación del firewall, se desarrolló un módulo en Python compatible con el controlador POX. Este módulo lee un archivo de configuración en formato JSON (**rules.json**) que contiene las reglas de filtrado a aplicar. El firewall se instala únicamente en el switch central de la topología (por ejemplo, **s3** en una topología de cinco switches), lo que permite controlar el tráfico que atraviesa el núcleo de la red.

El funcionamiento del firewall es el siguiente:

- Al iniciarse el controlador POX con el módulo de firewall, se cargan las reglas definidas en el archivo de configuración.
- Cuando el switch designado (por ejemplo, **s3**) establece conexión con el controlador, el módulo instala las reglas de filtrado como flujos en la tabla del switch.
- Cada regla puede especificar campos como dirección MAC, dirección IP, protocolo, puerto de origen o destino, entre otros. Si un paquete coincide con una regla de tipo **drop**, el switch descarta el tráfico correspondiente.
- El resto del tráfico no afectado por las reglas explícitas sigue el comportamiento por defecto del controlador (por ejemplo, reenvío normal o aprendizaje).

2.2.1. Reglas de filtrado aplicadas en el firewall

Para este trabajo se definieron un conjunto de reglas de firewall que buscan ilustrar distintos escenarios de filtrado de tráfico en la red. Las reglas se encuentran en el archivo `rules.json` y son las siguientes:

- **Bloqueo de tráfico entre h1 y h3:** Se bloquea todo el tráfico cuya dirección MAC de origen sea la de `h1` (00:00:00:00:01:01) y la de destino sea la de `h3` (00:00:00:00:01:03), y viceversa. Esto impide la comunicación directa entre estos dos hosts.
- **Bloqueo de tráfico UDP desde h1 hacia cualquier host en el puerto 5001:** Se bloquea todo paquete UDP (`nw_proto: 17`) proveniente de `h1` cuyo puerto de destino sea el 5001. Esta regla permite ilustrar el filtrado por protocolo y puerto.
- **Bloqueo de tráfico TCP y UDP dirigido al puerto 80:** Se bloquea todo tráfico TCP (`nw_proto: 6`) y UDP (`nw_proto: 17`) cuyo puerto de destino sea el 80, independientemente del host de origen o destino. Esto simula el bloqueo de servicios típicos como HTTP.

A continuación se muestra el contenido del archivo `rules.json` utilizado:

```
[
  {
    "mac_src": "00:00:00:00:01:01",
    "mac_dst": "00:00:00:00:01:03",
    "action": "drop"
  },
  {
    "mac_src": "00:00:00:00:01:03",
    "mac_dst": "00:00:00:00:01:01",
    "action": "drop"
  },
  {
    "mac_src": "00:00:00:00:01:01",
    "tp_dst": 5001,
    "nw_proto": 17,
    "action": "drop"
  },
  {
    "tp_dst": 80,
    "nw_proto": 6,
    "action": "drop"
  },
  {
    "tp_dst": 80,
    "nw_proto": 17,
    "action": "drop"
  }
]
```

Estas reglas permiten observar el impacto del filtrado selectivo sobre la conectividad y los servicios disponibles entre los hosts de la topología.

Campos disponibles en las reglas de flujo

Campo	Tipo	Descripción
dl_src	EthAddr	Dirección MAC de origen
dl_dst	EthAddr	Dirección MAC de destino
dl_type	int (hex)	Tipo de protocolo de capa 2 (por ejemplo, 0x0800 para IPv4, 0x0806 para ARP)
dl_vlan	int	ID de VLAN (red virtual local)
dl_vlan_pcp	int	Prioridad de VLAN
nw_src	IPAddr o IPAddr("x.x.x.x/x")	Dirección IP de origen
nw_dst	IPAddr	Dirección IP de destino
nw_proto	int	Protocolo de capa 3 (1 = ICMP, 6 = TCP, 17 = UDP)
nw_tos	int (0-255)	Tipo de servicio (ToS, usado para priorizar paquetes IP)
tp_src	int	Puerto de origen (TCP o UDP)
tp_dst	int	Puerto de destino (TCP o UDP)
in_port	int	Puerto de entrada en el switch (dispositivo que reenvía paquetes entre dispositivos en una LAN)

En este trabajo, se utilizó un conjunto reducido de campos en formato simplificado, como se observa en el archivo `rules.json`. Para que las reglas sean compatibles con el controlador, se aplicó un mapeo de campos como se detalla a continuación:

```
FIELD_MAP = {
    "mac_src": "dl_src",          # Dirección MAC de origen
    "mac_dst": "dl_dst",          # Dirección MAC de destino
    "dl_type": "dl_type",         # Tipo de protocolo de enlace de datos
    "dl_vlan": "dl_vlan",         # ID de VLAN
    "dl_vlan_pcp": "dl_vlan_pcp", # Prioridad de VLAN
    "nw_src": "nw_src",           # IP de origen
    "nw_dst": "nw_dst",           # IP de destino
    "nw_proto": "nw_proto",       # Protocolo de red
    "nw_tos": "nw_tos",           # Tipo de servicio
    "tp_src": "tp_src",           # Puerto de origen
    "tp_dst": "tp_dst",           # Puerto de destino
    "in_port": "in_port"         # Puerto de entrada del switch
}
```

Este mapeo permite traducir las claves utilizadas en el archivo JSON a los campos que el controlador espera internamente, facilitando la definición de reglas en un formato más amigable y legible.

2.2.2. Evidencia de instalación de reglas en el switch central

A continuación se presenta un extracto del log generado por el controlador POX al momento de instalar las reglas de firewall. En este log se puede observar que las reglas se aplican específicamente sobre el switch 3, que es el nodo central de la topología:

```
1 [openflow.of_01] [00-00-00-00-00-03 4] connected
2 [openflow.discovery] Installing flow for 00-00-00-00-00-03
3 [(...).firewall] Switch 3 has connected
4 [(...).firewall] Installing DROP rule: {
5   'mac_src': EthAddr('00:00:00:00:01:01'),
6   'mac_dst': EthAddr('00:00:00:00:01:03'),
7   'action': 'drop'
8 }
9 [(...).firewall] Installing DROP rule: {
10  'mac_src': EthAddr('00:00:00:00:01:03'),
11  'mac_dst': EthAddr('00:00:00:00:01:01'),
12  'action': 'drop'
```

```
13 }
14 [(...).firewall] Installing DROP rule: {
15   'mac_src': EthAddr('00:00:00:00:01:01'),
16   'tp_dst': 5001,
17   'nw_proto': 17,
18   'action': 'drop'
19 }
20 [(...).firewall] Installing DROP rule: {
21   'tp_dst': 80,
22   'nw_proto': 6,
23   'action': 'drop'
24 }
25 [(...).firewall] Installing DROP rule: {
26   'tp_dst': 80,
27   'nw_proto': 17,
28   'action': 'drop'
29 }
30 [(...).firewall] Installed firewall rules on switch 3
31 [forwarding.l2_learning] Connection [00-00-00-00-00-03 4]
```

Este log evidencia que todas las reglas definidas en el archivo de configuración fueron correctamente instaladas en el switch 3 al momento de su conexión al controlador, asegurando así el filtrado centralizado del tráfico según lo planificado.

Esta arquitectura permite modificar las reglas de filtrado de manera sencilla, editando el archivo `rules.json` y reiniciando el controlador. Además, al centralizar el filtrado en un único switch, se facilita el análisis y la depuración del comportamiento del firewall, permitiendo observar de manera clara el impacto de las reglas sobre el tráfico entre los distintos hosts. Como mencionamos anteriormente, el análisis detallado de la aplicación y los efectos de estas reglas se presentará en secciones posteriores del informe.

3. Pruebas

Para el desarrollo de las pruebas y la validación del comportamiento del firewall, se utilizó Wireshark para capturar y analizar el tráfico de red. Se configuró para capturar el tráfico en las interfaces `s1-eth1`, `s1-eth2`, `s5-eth1` y `s5-eth2`, lo que permite observar el tráfico antes y después de atravesar el switch central donde se encuentra implementado el firewall.

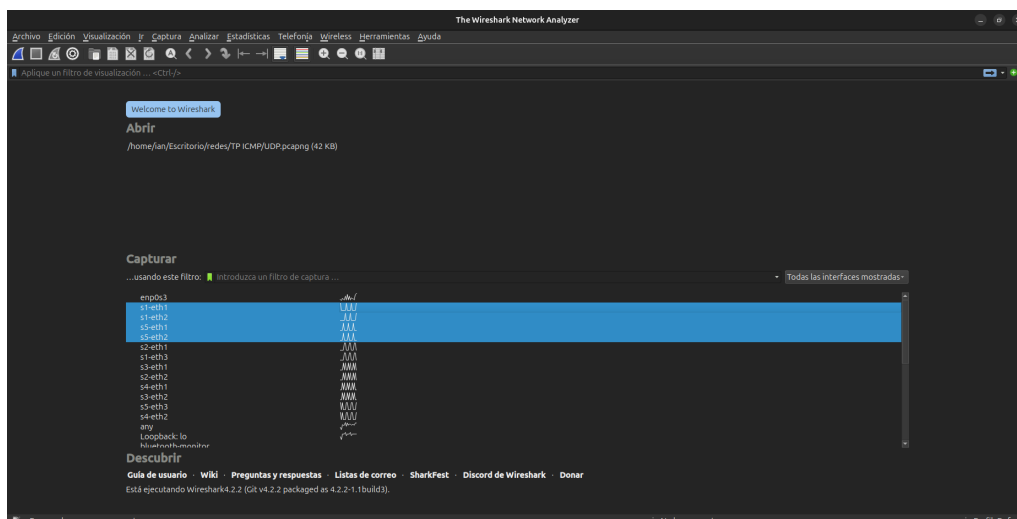


Figura 2: Configuración de captura de tráfico desde los switches extremos (s1 y s5)

Esta metodología de captura en los extremos resulta especialmente efectiva para demostrar el funcionamiento del firewall, ya que permite:

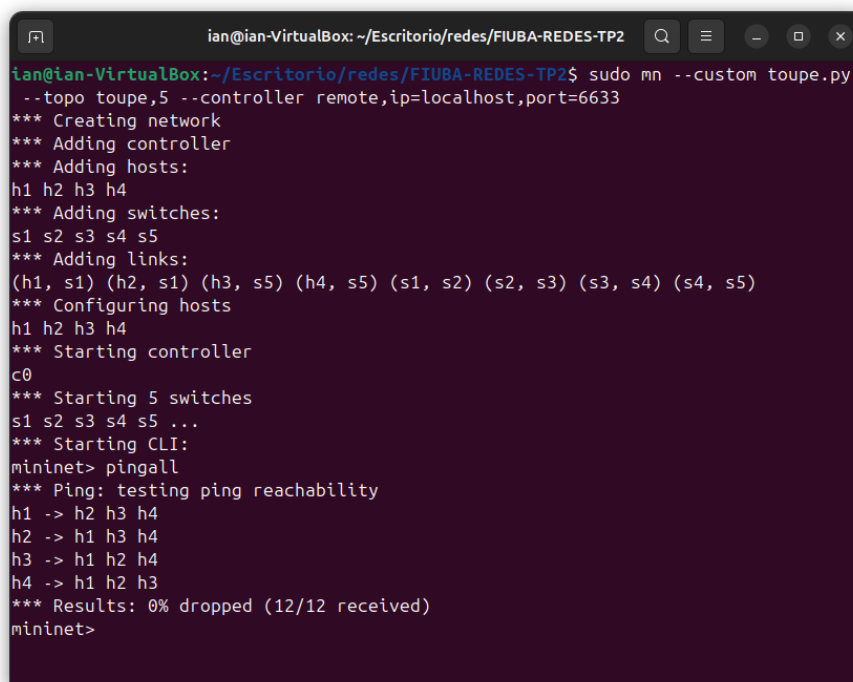
- **Verificar el origen del tráfico:** Las capturas en s1 muestran todo el tráfico generado por

los hosts h1 y h2, confirmando que los paquetes efectivamente se envían desde los hosts origen.

- **Validar el filtrado:** Las capturas en s5 revelan qué tráfico logra atravesar exitosamente toda la cadena de switches, incluyendo el firewall en s3.
- **Identificar bloqueos:** La ausencia de ciertos paquetes en s5 que sí aparecen en s1 confirma que el firewall está descartando el tráfico según las reglas configuradas.
- **Analizar el comportamiento bidireccional:** Se puede observar tanto el tráfico que va de s1 hacia s5 como el que retorna en sentido contrario.

3.1. Prueba 0: Conectividad básica sin firewall

Para establecer una línea base, se realizó una primera prueba de conectividad entre todos los hosts sin la aplicación de reglas de firewall. Esta prueba permite verificar que la topología funciona correctamente y que existe conectividad completa entre todos los nodos. Para ello, se utilizó `pingall`, una herramienta de mininet que simplemente hace ping entre todos los hosts de la topología.



```

ian@ian-VirtualBox: ~/Escritorio/redes/FIUBA-REDES-TP2
ian@ian-VirtualBox:~/Escritorio/redes/FIUBA-REDES-TP2$ sudo mn --custom toupe.py
--topo toupe,5 --controller remote,ip=localhost,port=6633
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4 s5
*** Adding links:
(h1, s1) (h2, s1) (h3, s5) (h4, s5) (s1, s2) (s2, s3) (s3, s4) (s4, s5)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
mininet>

```

Figura 3: Prueba de conectividad básica mediante ping entre todos los hosts

Como se observa en la figura anterior, todos los hosts pueden comunicarse exitosamente entre sí. Los comandos `ping` desde h1 hacia h2, h3 y h4 obtienen respuesta, confirmando que la topología está correctamente configurada y que el tráfico puede atravesar la cadena de switches sin restricciones.

*4 interfaces

Archivo Edición Visualización Ir Captura Analizar Estadísticas Telefonía Wireless Herramientas Ayuda

icmp

No.	Time	Source	Destination	Protocol	Length	Info
9	9.197435659	10.0.0.1	10.0.0.2	ICMP	98	s1-eth1
11	9.203221361	10.0.0.1	10.0.0.3	ICMP	98	s1-eth1
12	9.208683345	10.0.0.1	10.0.0.3	ICMP	98	s5-eth1
17	9.217625008	10.0.0.1	10.0.0.4	ICMP	98	s5-eth2
21	9.198356462	10.0.0.1	10.0.0.2	ICMP	98	s1-eth2
24	9.228847203	10.0.0.1	10.0.0.2	ICMP	98	s1-eth2
25	9.252397778	10.0.0.1	10.0.0.3	ICMP	98	s5-eth1
35	9.213960245	10.0.0.1	10.0.0.4	ICMP	98	s1-eth1
38	9.228181714	10.0.0.1	10.0.0.2	ICMP	98	s1-eth1
40	9.249233551	10.0.0.1	10.0.0.3	ICMP	98	s1-eth1
42	9.276009633	10.0.0.1	10.0.0.4	ICMP	98	s1-eth1
52	9.279132947	10.0.0.1	10.0.0.4	ICMP	98	s5-eth2
10	9.200441436	10.0.0.2	10.0.0.1	ICMP	98	s1-eth1
14	9.232683729	10.0.0.2	10.0.0.3	ICMP	98	s5-eth1
19	9.239660171	10.0.0.2	10.0.0.4	ICMP	98	s5-eth2
22	9.198368955	10.0.0.2	10.0.0.1	ICMP	98	s1-eth2
23	9.227144059	10.0.0.2	10.0.0.1	ICMP	98	s1-eth2
27	9.260420111	10.0.0.2	10.0.0.3	ICMP	98	s5-eth1
28	9.229935204	10.0.0.2	10.0.0.3	ICMP	98	s1-eth2
30	9.236723508	10.0.0.2	10.0.0.4	ICMP	98	s1-eth2
33	9.256869885	10.0.0.2	10.0.0.3	ICMP	98	s1-eth2
37	9.228168879	10.0.0.2	10.0.0.1	ICMP	98	s1-eth1
48	9.283228801	10.0.0.2	10.0.0.4	ICMP	98	s1-eth2
54	9.287803733	10.0.0.2	10.0.0.4	ICMP	98	s5-eth2
13	9.208697306	10.0.0.3	10.0.0.1	ICMP	98	s5-eth1
15	9.232695393	10.0.0.3	10.0.0.2	ICMP	98	s5-eth1
16	9.245789937	10.0.0.3	10.0.0.1	ICMP	98	s5-eth1
26	9.253612254	10.0.0.3	10.0.0.2	ICMP	98	s5-eth1
29	9.235453808	10.0.0.3	10.0.0.2	ICMP	98	s1-eth2
32	9.256856780	10.0.0.3	10.0.0.2	ICMP	98	s1-eth2
34	9.212471628	10.0.0.3	10.0.0.1	ICMP	98	s1-eth1
39	9.249222055	10.0.0.3	10.0.0.1	ICMP	98	s1-eth1
43	9.264213918	10.0.0.3	10.0.0.4	ICMP	98	s5-eth1
46	9.289989227	10.0.0.3	10.0.0.4	ICMP	98	s5-eth1
49	9.265881787	10.0.0.3	10.0.0.4	ICMP	98	s5-eth2
56	9.290661306	10.0.0.3	10.0.0.4	ICMP	98	s5-eth2
18	9.217639418	10.0.0.4	10.0.0.1	ICMP	98	s5-eth2
20	9.239673353	10.0.0.4	10.0.0.2	ICMP	98	s5-eth2
31	9.243450637	10.0.0.4	10.0.0.2	ICMP	98	s1-eth2
36	9.224112560	10.0.0.4	10.0.0.1	ICMP	98	s1-eth1
41	9.275974258	10.0.0.4	10.0.0.1	ICMP	98	s1-eth1
44	9.267278350	10.0.0.4	10.0.0.3	ICMP	98	s5-eth1
45	9.289977049	10.0.0.4	10.0.0.3	ICMP	98	s5-eth1
47	9.283215558	10.0.0.4	10.0.0.2	ICMP	98	s1-eth2
50	9.265912185	10.0.0.4	10.0.0.3	ICMP	98	s5-eth2
51	9.270522200	10.0.0.4	10.0.0.1	ICMP	98	s5-eth2
53	9.280329203	10.0.0.4	10.0.0.2	ICMP	98	s5-eth2
55	9.289344270	10.0.0.4	10.0.0.3	ICMP	98	s5-eth2

Internet Control Message Protocol: Protocol

Figura 4: Captura de Wireshark en los switches 1 y 5 durante la prueba de conectividad básica

Las capturas de Wireshark en los extremos de la topología (switch 1 y switch 5) muestran el tráfico ICMP correspondiente a los pings, incluyendo tanto los paquetes de solicitud (Echo Request) como las respuestas (Echo Reply). Este comportamiento confirma que el flujo de datos atraviesa correctamente toda la cadena de switches.

3.2. Prueba 1: Bloqueo de comunicación entre h1 y h3

Una vez aplicadas las reglas de firewall, se procedió a verificar el bloqueo bidireccional de la comunicación entre los hosts h1 y h3. Esta regla está definida en el archivo de configuración mediante el filtrado por direcciones MAC específicas (00:00:00:00:01:01 y 00:00:00:00:01:03). Para verificar completamente el bloqueo, se realizaron pruebas tanto con protocolo TCP como UDP utilizando la herramienta `iperf`, estableciendo alternativamente cada host como cliente y servidor.

3.2.1. Prueba TCP: h1 como cliente, h3 como servidor

En esta primera configuración, se estableció h3 como servidor TCP en el puerto por defecto de `iperf` (5001) y se intentó conectar desde h1 como cliente.

```

"host:h3"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -
s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----

"host:h1"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -
c 10.0.0.3
-----
Client connecting to 10.0.0.3, TCP port 5001
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 5001
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2#

```

Figura 5: Intento de conexión TCP desde h1 hacia h3 con firewall activo

En las terminales de `xterm` se observa que el servidor en `h3` queda esperando conexiones, mientras que el cliente en `h1` no logra establecer la conexión TCP. El cliente reporta errores de timeout o conexión rechazada, confirmando que el firewall está bloqueando efectivamente esta comunicación.

No.	Time	Source	Destination	Protocol	Length	Interfaz	Info
7.3.832	10.0.0.1	10.0.0.3	TCP	74	si-eth1	50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077013170 TSecr=0 WS=512	
8.4.862	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077014202 TSecr=0 WS=512	
9.5.886	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077015226 TSecr=0 WS=512	
10.6.910	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077016250 TSecr=0 WS=512	
12.7.934	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077017274 TSecr=0 WS=512	
16.8.958	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077018298 TSecr=0 WS=512	
17.11.00	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077019322 TSecr=0 WS=512	
22.15.00	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077020346 TSecr=0 WS=512	
35.23.55	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077031866 TSecr=0 WS=512	
49.39.93	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077042506 TSecr=0 WS=512	
88.72.19	10.0.0.1	10.0.0.3	TCP	74	si-eth1	[TCP Retransmission] 50006 - 5001 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM TSval=3077053030 TSecr=0 WS=512	

Figura 6: Captura de Wireshark durante intento de conexión TCP de h1 a h3

Las capturas de Wireshark revelan que los paquetes TCP SYN originados desde `h1` son enviados pero no llegan a `h3`, siendo descartados por el firewall en el switch central. No se observan respuestas SYN-ACK ni establecimiento de conexión TCP, confirmando el bloqueo efectivo.

3.2.2. Prueba TCP: h3 como cliente, h1 como servidor

Para verificar la bidireccionalidad del bloqueo, se invirtieron los roles: `h1` como servidor TCP y `h3` como cliente.

```

"host: h3"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -c 10.0.0.1
-----
Client connecting to 10.0.0.1, TCP port 5001
TCP window size: 85.3 KByte (default)
-----
tcp connect failed: Connection timed out
[ 1] local 0.0.0.0 port 0 connected with 10.0.0.1 port 5001
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2#

"host: h1"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -s
-----
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
-----

```

Figura 7: Intento de conexión TCP desde h3 hacia h1 con firewall activo

Los resultados son consistentes con la prueba anterior: el servidor en h1 permanece en estado de escucha mientras que el cliente en h3 no puede establecer la conexión. Esto confirma que el bloqueo opera en ambas direcciones, tal como está configurado en las reglas del firewall.

No.	Time	Source	Destination	Protocol	Length	Interface	Info
0.2.11.0	10.0.0.3	10.0.0.1	TCP	74	eth1	4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115597172 TSecr=0 WS=512	
0.3.123.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115597172 TSecr=0 WS=512	
7.4.147.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115581749 TSecr=0 WS=512	
8.5.171.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115581749 TSecr=0 WS=512	
12.6.195.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115581749 TSecr=0 WS=512	
14.7.213.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115581749 TSecr=0 WS=512	
15.9.267.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115586863 TSecr=0 WS=512	
20.11.291.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115586863 TSecr=0 WS=512	
37.21.68.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115592979 TSecr=0 WS=512	
50.38.06.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115616540 TSecr=0 WS=512	
70.70.32.0	10.0.0.3	10.0.0.1	TCP	74	eth1	[TCP Retransmission] 4584 - 5801 [SYN] Seq=0 Win=0 Len=0 MSS=1460 SACK_PERM TSval=115616549 TSecr=0 WS=512	

Figura 8: Captura de Wireshark durante intento de conexión TCP de h3 a h1

3.2.3. Prueba UDP: h1 como cliente, h3 como servidor

Para completar la verificación, se realizaron pruebas con protocolo UDP. Se configuró h3 como servidor UDP y h1 como cliente.

```

"host: h3"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -
s -u
-----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----

"host: h1"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -
c 10.0.0.3 -u
-----
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 1] local 10.0.0.1 port 32956 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0152 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2#

```

Figura 9: Intento de transmisión UDP desde h1 hacia h3 con firewall activo

En las terminales se observa que el cliente `iperf` en `h1` reporta el envío de datos UDP, pero el servidor en `h3` no recibe ningún tráfico. El reporte del cliente muestra pérdida del 100 % de los paquetes, indicando que todos los datagramas UDP están siendo bloqueados por el firewall.

No.	Time	Source	Destination	Protocol	Length	Interfaz	Info
1	0.000...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
2	0.011...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
3	0.011...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
4	0.022...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
5	0.033...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
6	0.044...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
7	0.056...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
8	0.067...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
9	0.078...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
10	0.089...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
11	0.101...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
12	0.112...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
13	0.123...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
14	0.134...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
15	0.145...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
16	0.157...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
17	0.168...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
18	0.179...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
19	0.190...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
20	0.202...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
21	0.213...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
22	0.224...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
23	0.235...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
24	0.247...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
25	0.258...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
26	0.269...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
27	0.280...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
28	0.291...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
29	0.302...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470
30	0.314...	10.0.0.1	10.0.0.3	UDP	1512	s1-eth1	40735 → 5001 Len=1470

Figura 10: Captura de Wireshark durante transmisión UDP bloqueada de h1 a h3

3.2.4. Prueba UDP: h3 como cliente, h1 como servidor

Finalmente, se invirtieron los roles para UDP: `h1` como servidor y `h3` como cliente.

```

"host: h3"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -c 10.0.0.1 -u
-----
Client connecting to 10.0.0.1, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 1] local 10.0.0.3 port 34445 connected with 10.0.0.1 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0153 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2#

"host: h1"
root@PC-JUMAS-LINUX:/home/jumas/Documentos/FIUBA/REDES/FIUBA-REDES-TP2# iperf -s -u
-----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----

```

Figura 11: Intento de transmisión UDP desde h3 hacia h1 con firewall activo

Los resultados confirman nuevamente el bloqueo bidireccional: el servidor UDP en h1 no recibe ningún dato del cliente en h3, con pérdida total de paquetes reportada por el cliente.

udp							
No.	Time	Source	Destination	Protocol	Length	Interfaz	Info
37	42.04...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
38	42.05...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
39	42.05...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
40	42.06...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
41	42.07...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
42	42.09...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
43	42.10...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
44	42.11...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
45	42.12...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
46	42.13...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
47	42.14...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
48	42.15...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
49	42.16...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
50	42.18...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
51	42.19...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
52	42.20...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
53	42.21...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
54	42.22...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
55	42.23...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
56	42.24...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
57	42.25...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
58	42.27...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
59	42.28...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
60	42.29...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
61	42.30...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
62	42.31...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470
63	42.32...	10.0.0.3	10.0.0.1	UDP	1512	s5-eth1	50215 → 5001 Len=1470

Figura 12: Captura de Wireshark durante transmisión UDP bloqueada de h3 a h1

3.2.5. Análisis de resultados

Las pruebas exhaustivas con ambos protocolos (TCP y UDP) y en ambas direcciones demuestran de manera concluyente que:

- El firewall bloquea efectivamente toda comunicación entre h1 y h3, independientemente del protocolo utilizado.
- El bloqueo es bidireccional, funcionando tanto cuando h1 intenta conectar a h3 como viceversa.

- Las reglas de filtrado por dirección MAC operan correctamente a nivel de enlace de datos, impidiendo que cualquier tráfico entre estos hosts atraviese el switch central.
- No se observó ningún tipo de comunicación exitosa entre ambos hosts durante todas las pruebas realizadas.

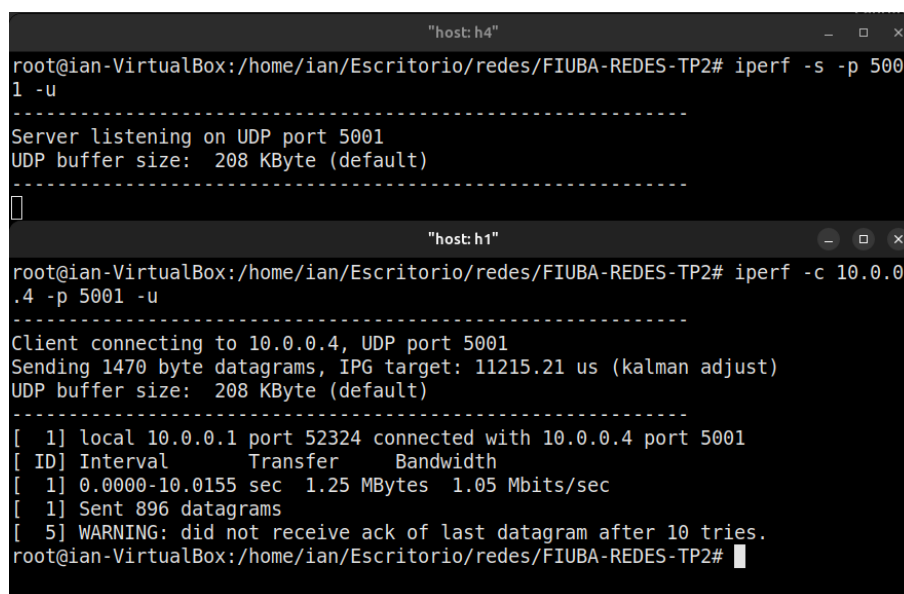
Este comportamiento confirma que las reglas de firewall definidas en el archivo `rules.json` se están aplicando correctamente en el switch central, cumpliendo con el objetivo de segmentar la comunicación entre hosts específicos de la red.

3.3. Prueba 2: Bloqueo de tráfico UDP en puerto 5001 desde h1

Esta prueba verifica la regla específica que bloquea el tráfico UDP desde `h1` hacia cualquier destino en el puerto 5001. Según las reglas definidas en el archivo de configuración, solo el tráfico UDP originado desde `h1` (`mac_src: "00:00:00:00:01:01"`) con destino al puerto 5001 debe ser bloqueado, mientras que el mismo tipo de tráfico desde otros hosts debe ser permitido.

3.3.1. Tráfico UDP bloqueado: h1 hacia h4 en puerto 5001

Se configuró `h4` como servidor UDP en el puerto 5001 e `h1` como cliente para verificar el bloqueo específico de esta regla.



```
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -s -p 5001 -u
-----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----
[ ]

root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -c 10.0.0.4 -p 5001 -u
-----
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 52324 connected with 10.0.0.4 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0155 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2#
```

Figura 13: Intento de envío de tráfico UDP desde h1 hacia h4 en puerto 5001

En las terminales de `xterm` se observa que el cliente `iperf` en `h1` reporta el envío de datos UDP hacia `h4`, pero el servidor en `h4` no recibe ningún tráfico. El reporte muestra una pérdida del 100 % de los paquetes, confirmando que el firewall está bloqueando efectivamente este tráfico específico basado en la dirección MAC de origen y el puerto de destino.

udp								
No.	Source	Destination	Protocol	Length	Info	Interfaz	src_port	dst_port
1139	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1140	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1141	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1142	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1143	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1144	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1145	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1146	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1147	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1148	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1149	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1150	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1151	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1152	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1153	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1154	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1155	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1157	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1158	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1159	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1160	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1161	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1162	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1163	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1164	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1165	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1166	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1167	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1168	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1169	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1170	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1171	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1172	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1173	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001
1174	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	52324	5001

Figura 14: Captura de Wireshark mostrando tráfico UDP bloqueado desde h1 al puerto 5001

Las capturas de Wireshark confirman que los paquetes UDP originados desde h1 con destino al puerto 5001 son enviados desde el host origen pero no llegan a su destino, siendo descartados por el firewall en el switch central. No se observa tráfico UDP correspondiente en el extremo de destino.

3.3.2. Tráfico UDP permitido: h2 hacia h4 en puerto 5001

Para verificar que la regla es específica solo para h1, se realizó la misma prueba pero utilizando h2 como cliente UDP hacia h4 en el mismo puerto 5001.

```

"host: h4"
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -s -p 5001 -u
-----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----
[ 1] local 10.0.0.4 port 5001 connected with 10.0.0.2 port 38906
[ ID] Interval      Transfer    Bandwidth      Jitter  Lost/Total Datagrams
[ 1] 0.0000-9.9779 sec 1.25 MBytes 1.05 Mbits/sec 0.120 ms 0/895 (0%)
[ 1] 0.0000-9.9779 sec 1 datagrams received out-of-order

"host: h2"
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -c 10.0.0.4 -p 5001 -u
-----
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 10.0.0.2 port 38906 connected with 10.0.0.4 port 5001
[ ID] Interval      Transfer    Bandwidth      Jitter  Lost/Total Datagrams
[ 1] 0.0000-10.0155 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 1] Server Report:
[ ID] Interval      Transfer    Bandwidth      Jitter  Lost/Total Datagrams
[ 1] 0.0000-9.9779 sec 1.25 MBytes 1.05 Mbits/sec 0.120 ms 0/895 (0%)
[ 1] 0.0000-9.9779 sec 1 datagrams received out-of-order
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2#

```

Figura 15: Envío exitoso de tráfico UDP desde h2 hacia h4 en puerto 5001

En contraste con la prueba anterior, las terminales muestran que el tráfico UDP desde h2 hacia h4 en el puerto 5001 fluye normalmente. El servidor en h4 recibe correctamente los datos enviados por el cliente en h2, mostrando estadísticas de transferencia exitosa sin pérdida de paquetes.

udp								
No.	Source	Destination	Protocol	Length	Info	Interfaz	src_port	dst_port
1782	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1783	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1784	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1785	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1786	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1787	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1788	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1789	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1790	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1791	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1792	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1793	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1794	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1795	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1796	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1797	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1798	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1799	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1800	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1801	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1802	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1803	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1804	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1805	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1806	10.0.0.2	10.0.0.4	UDP	1512	s1-eth2	s1-eth2	38906	5001
1807	10.0.0.4	10.0.0.2	UDP	170	s1-eth2	s1-eth2	5001	38906
1808	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1809	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1810	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1811	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1812	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1813	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1814	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1815	10.0.0.2	10.0.0.4	UDP	1512	s5-eth2	s5-eth2	38906	5001
1816	10.0.0.4	10.0.0.2	UDP	170	s5-eth2	s5-eth2	5001	38906

Figura 16: Captura de Wireshark mostrando tráfico UDP permitido desde h2 al puerto 5001

Las capturas de Wireshark evidencian el flujo normal de paquetes UDP desde **h2** hacia **h4** en el puerto 5001. Se observa el tráfico completo atravesando la red sin interferencias del firewall, confirmando que la regla de bloqueo es específica únicamente para el tráfico originado desde **h1**.

3.3.3. Análisis de resultados

Esta prueba demuestra la precisión y selectividad de las reglas de firewall implementadas:

- La regla que bloquea tráfico UDP desde **h1** al puerto 5001 opera correctamente, filtrando específicamente por dirección MAC de origen (00:00:00:00:01:01), protocolo (UDP, `nw_proto: 17`) y puerto de destino (`tp_dst: 5001`).
- El mismo tipo de tráfico desde otros hosts (**h2**) hacia el mismo puerto y protocolo no se ve afectado, confirmando que el filtrado es granular y no impacta comunicaciones no especificadas en las reglas.
- El firewall permite un control fino del tráfico, bloqueando comunicaciones específicas mientras mantiene la funcionalidad general de la red.
- La implementación distingue correctamente entre diferentes hosts origen, aplicando las reglas solo cuando se cumplen todos los criterios especificados.

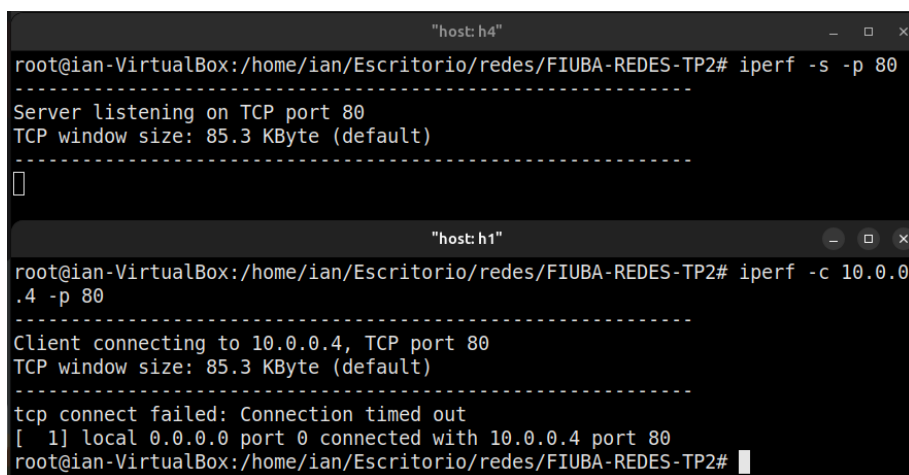
Este comportamiento valida que el sistema de firewall puede implementar políticas de seguridad complejas que afecten selectivamente a hosts específicos sin comprometer la conectividad global de la red.

3.4. Prueba 4: Bloqueo de tráfico HTTP (puerto 80) TCP y UDP

Para verificar las reglas que bloquean todo el tráfico hacia el puerto 80, tanto TCP como UDP, se realizaron pruebas exhaustivas utilizando la herramienta **iperf** para generar tráfico en ambos protocolos. Según las reglas definidas, todo el tráfico dirigido al puerto 80 debe ser bloqueado independientemente del host origen o protocolo utilizado.

3.4.1. Bloqueo de tráfico TCP en puerto 80

Se configuró un servidor TCP en el puerto 80 y se intentó establecer conexiones desde diferentes hosts para verificar el bloqueo efectivo de este protocolo.

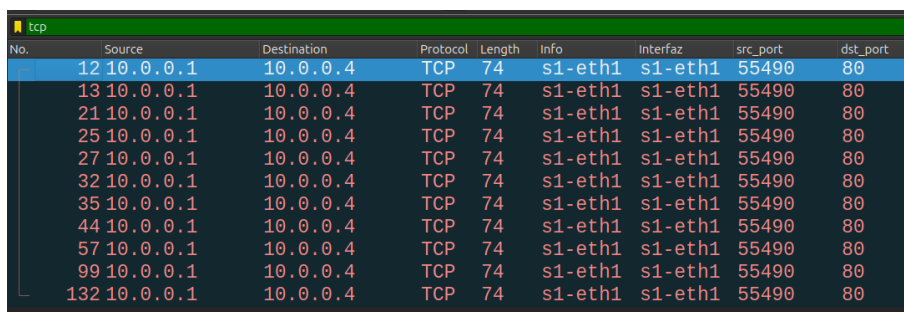


```
"host: h4"
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -s -p 80
-----
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
-----
[ ]

"host: h1"
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -c 10.0.0.4 -p 80
-----
Client connecting to 10.0.0.4, TCP port 80
TCP window size: 85.3 KByte (default)
-----
tcp connect failed: Connection timed out
[ 1] local 0.0.0.0 port 0 connected with 10.0.0.4 port 80
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2#
```

Figura 17: Intento de conexión TCP al puerto 80 con firewall activo

En las terminales de **xterm** se observa que los intentos de establecer conexiones TCP en el puerto 80 fallan sistemáticamente. El cliente **iperf** no logra conectarse al servidor, reportando errores de timeout o conexión rechazada. Esto confirma que el firewall está bloqueando efectivamente todo el tráfico TCP dirigido al puerto 80, independientemente del host origen.



The image shows a Wireshark packet capture titled 'tcp'. It displays a list of 12 TCP packets. All packets have a source IP of 10.0.0.1 and a destination IP of 10.0.0.4. They are all TCP SYN packets (Length 74) originating from s1-eth1 and destined for s1-eth1 on port 55490, with the destination port being 80. The packet numbers are 12, 13, 21, 25, 27, 32, 35, 44, 57, 99, and 132.

No.	Source	Destination	Protocol	Length	Info	Interfaz	src_port	dst_port
12	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
13	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
21	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
25	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
27	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
32	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
35	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
44	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
57	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
99	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80
132	10.0.0.1	10.0.0.4	TCP	74	s1-eth1 s1-eth1 55490 80	s1-eth1	55490	80

Figura 18: Captura de Wireshark mostrando bloqueo de tráfico TCP en puerto 80

Las capturas de Wireshark revelan que los paquetes TCP SYN destinados al puerto 80 son enviados desde los hosts origen pero no se observan en el destino, confirmando que están siendo descartados por el firewall en el switch central. No se registran respuestas TCP SYN-ACK ni establecimiento de conexiones TCP, lo que valida el correcto funcionamiento de la regla de bloqueo.

3.4.2. Bloqueo de tráfico UDP en puerto 80

Para completar la verificación, se realizaron pruebas similares utilizando protocolo UDP en el mismo puerto.



The image shows two terminal windows. The top window, titled 'host: h4', shows a server running 'iperf -s -p 80 -u' and listening on UDP port 80. The bottom window, titled 'host: h1', shows a client running 'iperf -c 10.0.0.4 -p 80 -u'. The client output shows it connected to 10.0.0.4 on port 80 and sent 896 datagrams. However, it received a warning: 'WARNING: did not receive ack of last datagram after 10 tries.' This indicates that the server is not receiving the datagrams due to the firewall.

```

root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -s -p 80 -u
Server listening on UDP port 80
UDP buffer size: 208 KByte (default)

root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2# iperf -c 10.0.0.4 -p 80 -u
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 10.0.0.1 port 51242 connected with 10.0.0.4 port 80
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0162 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@ian-VirtualBox:/home/ian/Escritorio/redes/FIUBA-REDES-TP2#

```

Figura 19: Intento de transmisión UDP al puerto 80 con firewall activo

Los resultados con UDP son consistentes con los obtenidos para TCP. El cliente **iperf** en modo UDP reporta el envío de datos al puerto 80, pero el servidor no recibe ningún tráfico. Las estadísticas muestran una pérdida del 100 % de los paquetes, confirmando que todos los datagramas UDP dirigidos al puerto 80 están siendo bloqueados por el firewall.

No.	Source	Destination	Protocol	Length	Info	Interfaz	src_port	dst_port
7	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
8	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
9	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
10	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
11	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
12	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
13	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
14	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
15	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
16	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
17	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
18	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
19	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
20	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
21	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
22	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
23	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
24	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
25	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
26	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
27	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
28	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
29	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
30	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
31	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
32	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
33	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
34	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
35	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
36	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
37	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
38	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
39	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
40	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80
41	10.0.0.1	10.0.0.4	UDP	1512	s1-eth1	s1-eth1	51242	80

Figura 20: Captura de Wireshark mostrando bloqueo de tráfico UDP en puerto 80

Las capturas de red confirman que los paquetes UDP destinados al puerto 80 tampoco llegan a su destino, siendo interceptados y descartados por el firewall. Este comportamiento demuestra que las reglas de bloqueo por puerto operan correctamente para ambos protocolos de transporte.

3.4.3. Análisis de resultados

Esta prueba demuestra de manera concluyente que:

- Las reglas de bloqueo por puerto funcionan correctamente para ambos protocolos de transporte (TCP y UDP).
- El filtrado por puerto opera independientemente del host origen, bloqueando todo el tráfico dirigido al puerto 80 sin importar desde qué host se origine.
- La prioridad de las reglas de firewall (prioridad 100) garantiza que tomen precedencia sobre las reglas de aprendizaje L2 normales.
- El bloqueo es efectivo y silencioso, descartando los paquetes sin generar respuestas de error que podrían revelar información sobre la infraestructura de red.
- El sistema mantiene la funcionalidad normal para todo el tráfico no afectado por las reglas específicas.

3.5. Análisis general de resultados

Los resultados de todas las pruebas realizadas demuestran que el firewall implementado funciona correctamente, aplicando de manera selectiva las reglas definidas en el archivo de configuración. Se observa que:

- Las reglas de bloqueo por dirección MAC funcionan bidireccionalmente, impidiendo la comunicación específica entre h1 y h3 para cualquier protocolo.

- El filtrado por protocolo y puerto opera efectivamente, bloqueando tráfico UDP específico desde h1 al puerto 5001 mientras permite el mismo tipo de tráfico desde otros hosts.
- Las reglas generales por puerto bloquean todo el tráfico dirigido al puerto 80, independientemente del host origen o protocolo utilizado.
- El tráfico no afectado por las reglas continúa fluyendo normalmente a través de la topología, manteniendo la conectividad general de la red.

La implementación del firewall en el switch central (s3) permite un control granular del tráfico que atraviesa el núcleo de la red, manteniendo la conectividad general mientras aplica políticas de seguridad específicas según los requisitos definidos. La arquitectura SDN facilita la implementación y modificación de estas políticas de manera centralizada y flexible.

4. Preguntas

4.1. ¿Cual es la diferencia entre un Switch y un router? ¿Que tienen en comun?

Un switch opera principalmente en la capa de enlace del modelo de capas de protocolos utilizados en la Internet. Su función principal es gestionar la comunicación entre dispositivos dentro de una red local (LAN). Para ello, utiliza direcciones MAC para identificar los dispositivos conectados y mantiene una tabla de direcciones que le permite reenviar los datos únicamente al puerto correspondiente, optimizando el tráfico interno y minimizando las colisiones. De esta forma, el switch asegura una transmisión eficiente de datos dentro de un mismo segmento de red, sin intervenir en la comunicación hacia redes externas.

Un router, por otro lado, se encuentra en la capa de red y su propósito es conectar distintas redes entre sí. Utiliza direcciones IP para tomar decisiones respecto a que ruta optima deben viajar los paquetes para que lleguen a su destino, incluso cuando atraviesan multiples redes. El router gestiona el tráfico entre redes separadas, como entre una red local y la Internet, consultando su tabla de enrutamiento para guiar los paquetes de manera eficiente.

En cuanto a las coincidencias, ambos dispositivos tienen en común el objetivo de facilitar el envío de datos y la interconexión de dispositivos, pero cada uno lo hace en niveles distintos: el switch a nivel de red local y direcciones físicas (MAC), y el router a nivel de inter-redes y direcciones lógicas (IP).

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

Un **switch convencional** es un dispositivo de red que opera típicamente en la capa de enlace, reenvía tramas (*frames*) basándose en direcciones MAC, y mantiene una tabla de direcciones construida mediante aprendizaje automático a partir del tráfico observado. Su lógica de operación está embebida en el firmware del dispositivo y responde a algoritmos fijos, que no pueden ser modificados directamente por el administrador.

En cambio, un **switch OpenFlow** está diseñado para operar bajo el paradigma de redes definidas por software. No toma decisiones de reenvío de forma autónoma, sino que depende de un *controlador externo* para instalar reglas en su *tabla de flujos*. Estas reglas pueden utilizar campos de cabeceras de múltiples capas (MAC, IP, puertos, protocolo, etc.) para definir flujos con granularidad fina.

La principal diferencia radica en la **programabilidad y flexibilidad**: mientras que los switches tradicionales operan con lógica cerrada, los switches OpenFlow permiten implementar políticas personalizadas (firewalls, NAT, balanceo de carga, etc.) de forma dinámica mediante la intervención directa del controlador. Sin embargo, esto también implica una dependencia funcional del controlador, ya que, sin él, el switch OpenFlow carece de comportamiento autónomo.

En resumen, un switch OpenFlow representa un dispositivo de forwarding *desacoplado de su control*, mientras que un switch convencional combina en un único equipo tanto la lógica de decisión como el reenvío de paquetes.

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

Si bien los switches OpenFlow ofrecen una arquitectura altamente flexible y programable para el control del tráfico, **no es viable reemplazar todos los routers de Internet por switches OpenFlow**, especialmente en escenarios de ruteo entre sistemas autónomos (inter-AS).

Los routers tradicionales, especialmente los que operan en la capa de red en los *puntos de intercambio de tráfico* (IXPs) o fronteras de sistemas autónomos, están diseñados para ejecutar protocolos complejos como BGP (Border Gateway Protocol), los cuales son esenciales para la operación global de Internet. Estos protocolos permiten que cada organización defina políticas de ruteo, gestione múltiples caminos redundantes y tome decisiones autónomas en caso de fallas.

Por otro lado, los switches OpenFlow fueron pensados principalmente para entornos controlados como centros de datos o redes corporativas, donde existe un único dominio de control centralizado. En tales contextos, el controlador SDN tiene visibilidad global y puede instalar reglas óptimas. Sin embargo, esta arquitectura no escala adecuadamente a nivel global, ya que:

- Requiere conectividad constante con un controlador, lo cual introduce un punto único de fallo.
- No implementa por defecto protocolos inter-AS como BGP.
- La cantidad de flujos individuales en Internet es inmensa, lo que podría sobrepasar la capacidad de almacenamiento las tablas de flujo.

Aunque OpenFlow y SDN son adecuados para reemplazar funciones internas de enrutamiento en redes privadas o centros de datos, **no son un reemplazo directo y completo para los routers tradicionales en la arquitectura global de Internet**. Ambos modelos coexisten en la práctica, siendo OpenFlow una herramienta complementaria para entornos controlados y de escala limitada.

5. Conclusiones

El desarrollo de este trabajo práctico nos brindó una visión general y práctica en el ámbito de las redes definidas por software (SDN), permitiendo comprender de manera completa los conceptos teóricos relacionados con OpenFlow, controladores SDN y la implementación de políticas de red centralizadas.

5.1. Aprendizajes sobre SDN y OpenFlow

La implementación del firewall mediante el controlador POX demostró las ventajas fundamentales del paradigma SDN: la separación entre el plano de control y el plano de datos permite una gestión centralizada y flexible de las políticas de red. A diferencia de los enfoques tradicionales donde cada dispositivo de red opera de manera independiente, SDN facilita la implementación de reglas complejas desde un punto central, simplificando la administración y permitiendo cambios dinámicos en el comportamiento de la red.

5.2. Experiencia con la API de POX

El trabajo con la API de POX presentó tanto aspectos positivos como dificultades. Por un lado, POX ofrece una abstracción clara y relativamente bien documentada de los conceptos de OpenFlow, lo cual facilita el desarrollo de aplicaciones SDN mediante Python. La estructura orientada a eventos del framework permite una programación intuitiva, donde los manejadores de eventos responden a conexiones de switches y a la llegada de paquetes.

Sin embargo, durante el desarrollo surgieron inconvenientes técnicos relacionados con la compatibilidad entre POX y Python 3. En particular, al ejecutar ciertos comandos desde Mininet se producían errores en la consola de POX asociados al análisis de paquetes DHCP y DNS. Estos errores estaban ligados a diferencias entre Python 2 y Python 3 en el manejo de tipos de datos ('str' y 'bytes'). Como solución, fue necesario buscar en foros e *issues* del repositorio de POX, donde se sugería modificar directamente partes del código fuente, como remover el uso de 'ord()' en ciertos métodos y agregar prefijos 'b' en operaciones con cadenas. Estas modificaciones permitieron continuar con las pruebas, aunque evidencian la falta de una versión completamente estable de POX para Python 3. Adjuntamos el enlace al *issue* correspondiente: <https://github.com/noxrepo/pox/issues/251>.

5.3. Desafíos del desarrollo del firewall

Uno de los puntos importantes fue establecer bien el orden de las reglas para que no haya conflictos con las reglas por defecto de aprendizaje L2. También fue importante comprobar que las reglas se aplicaran como esperábamos, para lo cual hicimos pruebas desde las terminales de los hosts y analizamos el tráfico con **Wireshark**.

5.4. Herramientas de análisis y validación

El uso combinado de herramientas como **iperf**, **ping**, **ovs-ofctl** y **Wireshark** demostró ser fundamental para la validación del comportamiento del firewall. Cada herramienta proporcionó perspectivas complementarias: **iperf** para verificar el bloqueo de protocolos específicos, **ping** para pruebas básicas de conectividad, **ovs-ofctl** para inspeccionar directamente las tablas de flujo, brindando estadísticas de puerto y tablas de switch, y **Wireshark** para análisis detallado del tráfico a nivel de paquetes.

5.5. Aplicabilidad en entornos reales

Los conceptos y técnicas desarrollados en este trabajo creemos que tienen aplicación directa en entornos de red reales. La capacidad de implementar firewalls centralizados, segmentación de red dinámica y políticas para mejorar la calidad del servicio mediante SDN representa ventajas tangibles para organizaciones que buscan mayor control y flexibilidad en su infraestructura de red.