

# Trabajo Práctico N°2

Introducción a los Sistemas Distribuidos [TA049]

## Integrantes:

- 108091, Martín Morilla
- 107552, Iñaki Llorens
- 108313, Rafael Ortigano

Fecha de entrega: 24 de Junio de 2025



# Índice

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Hipótesis y Suposiciones</b>	<b>2</b>
<b>3</b>	<b>Implementación</b>	<b>3</b>
3.1	Arquitectura General . . . . .	3
3.2	Topología . . . . .	3
3.3	Controlador Remoto: OpenFlow y POX . . . . .	4
3.4	Implementación del Firewall SDN . . . . .	4
3.5	Reglas de Firewall Implementadas . . . . .	5
<b>4</b>	<b>Ejecución del Programa</b>	<b>6</b>
<b>5</b>	<b>Tests y Resultados</b>	<b>6</b>
5.1	Pruebas de conectividad . . . . .	7
5.2	Pruebas de bloqueo con reglas de firewall . . . . .	7
5.3	Logs del controlador . . . . .	8
<b>6</b>	<b>Dificultades Encontradas</b>	<b>9</b>
6.1	Configuración Inicial del Entorno SDN . . . . .	9
6.2	Comprensión del Paradigma SDN . . . . .	9
6.3	Gestión de Tablas de Flujo OpenFlow . . . . .	10
<b>7</b>	<b>Preguntas a Responder</b>	<b>10</b>
<b>8</b>	<b>Conclusión</b>	<b>11</b>

## 1. Introducción

El presente trabajo práctico se centra en el estudio de las Redes definidas por Software (SDN) y su implementación mediante el protocolo OpenFlow. A través del desarrollo de una simulación controlada, se busca comprender los principios fundamentales de este paradigma y explorar su aplicabilidad en entornos reales. El trabajo se estructura en torno a los siguientes ejes:

- **Diseño e implementación de una topología dinámica:** Se desarrolló una topología parametrizable utilizando el simulador Mininet, que permite variar la cantidad de switches conectados formando una topología lineal donde en cada extremo se sitúan 2 hosts.
- **Controlador externo con funcionalidad de firewall:** Se empleó la API POX para programar un controlador que gestiona las reglas de flujo de los switches, simulando un firewall de capa de enlace con reglas definidas mediante un archivo JSON.
- **Separación del plano de control y de datos:** La solución implementada permite evidenciar cómo la centralización del control mejora la flexibilidad, la gestión y la adaptabilidad de la red.
- **Validación experimental:** Se realizaron pruebas funcionales utilizando herramientas como `iperf` y `Wireshark`, con el fin de verificar el correcto funcionamiento de la red y la efectividad del firewall.

## 2. Hipótesis y Suposiciones

- **Topología mínima:** Se asume que la red cuenta con al menos un switch, y que los nombres asignados siguen el formato `s1`, `s2`, etc., como en Mininet.
- **Estabilidad del controlador:** Se supone que POX corre de forma estable y compatible, permitiendo la carga dinámica de reglas mediante `rules.json`.
- **Funcionamiento esperado de las reglas:** Las reglas de firewall son aplicadas correctamente en el evento `ConnectionUp`, bloqueando el tráfico según las condiciones especificadas.
- **Comportamiento determinista:** La red responde de manera predecible a los eventos definidos, sin interferencias ni comportamiento no deseado.
- **Topología confiable y escalable:** Se espera que la red simulada en Mininet no experimente fallos, y que la cantidad de switches pueda escalar sin afectar el funcionamiento del sistema.

## 3. Implementación

### 3.1. Arquitectura General

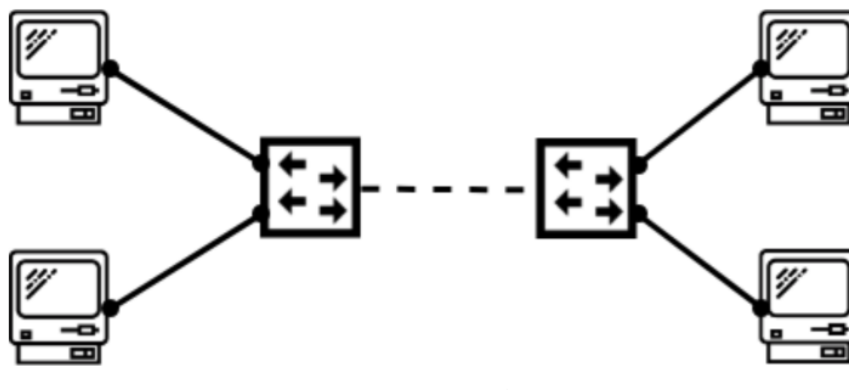


Figura 1: Topología

### 3.2. Topología

El archivo `linear-topology.py` define una topología de red basada en el paradigma de Redes Definidas por Software (SDN), utilizando Mininet para su implementación. La estructura propuesta representa una arquitectura lineal escalable, ideal para estudiar el comportamiento del tráfico en redes con múltiples saltos. A continuación, se detallan las características principales:

- **Número de switches:** La topología admite una cantidad variable de switches ( $n$ ), configurable. Esta parametrización permite experimentar con distintos tamaños de red.
- **Distribución de hosts:** Se crean cuatro hosts ( $h1$ ,  $h2$ ,  $h3$ ,  $h4$ ) con direcciones IP estáticas en la subred  $10.0.0.0/24$ .
  - $h1$  y  $h2$  se conectan al primer switch ( $s1$ ).
  - $h3$  y  $h4$  se conectan al último switch ( $s_n$ ).
- **Interconexión de switches:** Los switches se conectan en serie, formando una cadena lineal. Cada switch intermedio se comunica únicamente con su predecesor y sucesor, excepto los extremos, que tienen un único vecino.
- **Características de conectividad:**
  - El tráfico entre hosts del mismo extremo atraviesa un único switch, minimizando la latencia.
  - El tráfico entre extremos distintos atraviesa toda la cadena de switches, permitiendo que un switch intermedio (por ejemplo,  $s2$ ) funcione como punto de control de políticas de seguridad.
- **Validación de configuración:** La implementación incluye validaciones que aseguran que el número de switches sea mayor a cero.

Esta arquitectura resulta especialmente adecuada para estudiar:

- La separación entre el plano de control y el plano de datos.
- La aplicación de políticas de filtrado mediante un controlador externo.

### 3.3. Controlador Remoto: OpenFlow y POX

Para definir el comportamiento de los switches de la topología previamente descrita, se implementó un controlador SDN centralizado. Este controlador comunica a los switches OpenFlow qué reglas deben aplicar, delegando la lógica de control a un componente externo a la red de datos.

- **Protocolo OpenFlow:** OpenFlow es el protocolo de red que permite la separación entre el *plano de control* y el *plano de datos*. En este modelo:
  - El plano de control reside en un controlador externo.
  - El plano de datos queda limitado a reenviar paquetes siguiendo reglas impuestas por el controlador.

Esta arquitectura habilita un control centralizado y dinámico sobre la red, permitiendo modificar reglas de reenvío sin necesidad de reconfigurar individualmente cada dispositivo.

- **Framework POX:** La implementación del controlador se realizó utilizando POX, un framework en Python que facilita la creación de aplicaciones SDN mediante un enfoque basado en eventos. POX ofrece una interfaz simple para manejar eventos generados por los switches OpenFlow, tales como:
  - **ConnectionUp / ConnectionDown:** conexión y desconexión de switches.
  - **PacketIn / PacketOut:** llegada y envío de paquetes al controlador.
  - **FlowRemoved:** eliminación de reglas de flujo activas.
  - **PortStatus:** cambios en el estado de los puertos del switch.

El uso combinado de OpenFlow y POX permite definir políticas complejas de control de tráfico, incluyendo funciones como filtrado, redireccionamiento o análisis, adaptables en tiempo real a partir del tráfico observado.

### 3.4. Implementación del Firewall SDN

Un **firewall** es un componente esencial de seguridad en redes, encargado de filtrar el tráfico de acuerdo a un conjunto de reglas previamente definidas. Su función principal es decidir si se permite o se bloquea el paso de paquetes en función de características como dirección IP, puerto, o protocolo. En el contexto de redes definidas por software (SDN), esta lógica de filtrado puede ser implementada directamente desde el controlador central, lo cual ofrece mayor flexibilidad y dinamismo que las soluciones tradicionales.

En nuestro proyecto, el firewall fue implementado como una funcionalidad adicional dentro del controlador SDN basado en POX. La lógica de filtrado se encuentra contenida en el módulo `firewall.py`, el cual es responsable de instalar reglas específicas de bloqueo

en los switches de la red. Las reglas que definen qué tipo de tráfico debe bloquearse están externalizadas en un archivo `rules.json`, lo cual facilita su mantenimiento y extensión sin necesidad de modificar el código fuente.

- **Asociación con el switch:** El firewall puede ser configurado para aplicarse a un switch específico (por ejemplo, `s2`) o a todos los switches de la topología. Esto se logra al interpretar el nombre del switch desde su *dpid* (Datapath ID) y compararlo con el definido en la configuración.
- **Carga dinámica de reglas:** Las reglas se cargan al momento de iniciar el controlador y son aplicadas a cada switch (si no se provee un switch específico) durante el evento `ConnectionUp`, el cual indica que el switch ha sido conectado al controlador.
- **Aplicación de las reglas en los switches:** Una vez identificado el switch correspondiente, el controlador instala reglas de flujo (*flow rules*) directamente en el plano de datos del switch utilizando instrucciones OpenFlow. Estas reglas especifican qué paquetes deben ser descartados sin reenvío, lo que implementa de facto el comportamiento de firewall.
- **Independencia del tráfico general:** Para todo el tráfico que no coincide con las reglas de bloqueo, se delega el control al módulo de `switch_controller.py`, que maneja el aprendizaje de direcciones MAC (switch L2 learning), asegurando que el firewall no interfiere en la conectividad general de la red.

Esta implementación demuestra la potencia del enfoque SDN, al permitir gestionar políticas de seguridad de forma centralizada, programable y desacoplada del hardware. Además, la utilización de archivos externos para las reglas habilita la automatización y la integración con otras herramientas de gestión de red.

### 3.5. Reglas de Firewall Implementadas

Las reglas de filtrado del firewall fueron definidas en un archivo externo en formato JSON, lo que permite su modificación sin alterar el código del controlador. Cada regla especifica condiciones que deben cumplirse para que un paquete sea descartado, basándose en campos como protocolo, puerto de destino y dirección IP de origen.

#### Mecanismo de aplicación de reglas

En el modelo SDN con OpenFlow, las reglas de filtrado se traducen en entradas en la **tabla de flujos** de cada switch. Cada entrada define un patrón de coincidencia sobre determinados campos del encabezado del paquete (conocidos como *match fields*) y una acción a realizar si se cumple dicho patrón (por ejemplo, `DROP`).

Los switches OpenFlow soportan el análisis de múltiples campos para identificar un flujo, entre ellos:

- Dirección IP de origen y destino (`nw_src`, `nw_dst`)
- Protocolo de red (por ejemplo, TCP o UDP, mediante `nw_proto`)
- Puerto de origen y destino (`tp_src`, `tp_dst`)

El controlador POX, al cargar el archivo `rules.json`, traduce cada regla a una estructura OpenFlow válida, y la instala en los switches mediante mensajes del tipo `ofp_flow_mod` con acción `DROP`. Esto le indica al switch que debe descartar cualquier paquete que coincida con los criterios especificados.

## Ejemplos de reglas

- **Bloqueo por puerto destino:**

```
{"nw_proto": "TCP", "tp_dst": 80}
```

Esta regla bloquea todo el tráfico TCP dirigido al puerto 80 (HTTP), impidiendo conexiones a servidores web. Cuando un paquete TCP con destino 80 llega al switch, éste detecta que cumple la regla y lo descarta automáticamente.

- **Bloqueo por host y puerto en protocolo UDP:**

```
{"nw_proto": "UDP", "nw_src": "10.0.0.1", "tp_dst": 5001}
```

Esta regla bloquea paquetes UDP enviados por el host 10.0.0.1 con destino al puerto 5001. Es útil, por ejemplo, para limitar pruebas de rendimiento hechas con `iperf` en ese puerto específico.

Gracias a esta arquitectura, la lógica del firewall puede ser modificada dinámicamente desde el controlador, sin necesidad de tocar la configuración de los switches físicos o virtuales.

## 4. Ejecución del Programa

El proceso completo de ejecución del proyecto —incluyendo los comandos necesarios para inicializar la topología en Mininet, levantar el controlador POX con las funcionalidades correspondientes (switch y firewall), así como realizar pruebas de conectividad y filtrado de tráfico— se encuentra detallado en el archivo `README` del repositorio.

Allí se ofrece una guía paso a paso que permite replicar el entorno experimental y verificar el correcto funcionamiento de cada componente del sistema. Se recomienda seguir las instrucciones en el orden indicado para garantizar una correcta inicialización del entorno y una validación adecuada de las funcionalidades implementadas.

## 5. Tests y Resultados

Con el objetivo de verificar el correcto funcionamiento de las reglas del firewall y la conectividad general de la red, se realizaron diversas pruebas utilizando herramientas como `iperf`, `ping` y `Wireshark`. A su vez, se analizaron los logs generados por el controlador POX para observar la aplicación de las reglas y la gestión de eventos de red. Vamos a elegir una red con 3 switches, por lo que la siguiente figura representa nuestra topología:

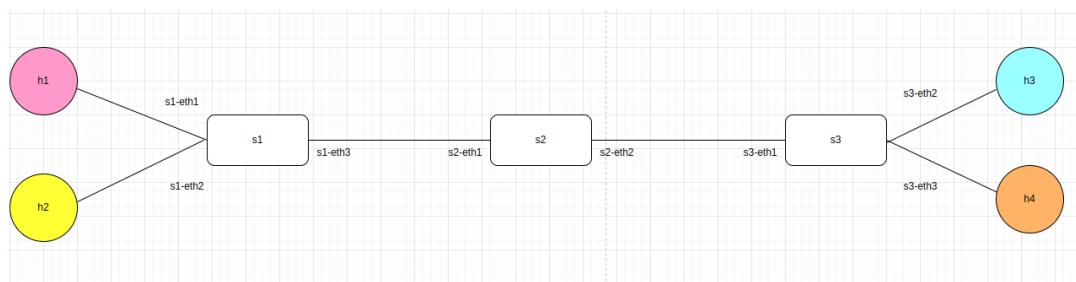


Figura 2: Topología lineal con 3 switches

## 5.1. Pruebas de conectividad

Inicialmente, se validó el comportamiento de la topología ejecutando el comando `pingall` desde el entorno de Mininet. Esta prueba confirmó que los hosts podían comunicarse correctamente en ausencia de reglas restrictivas.

```

2025-06-24 15:16:13 - main - INFO - IPs configuradas:
2025-06-24 15:16:13 - main - INFO - h1: 10.0.0.1
2025-06-24 15:16:13 - main - INFO - h2: 10.0.0.2
2025-06-24 15:16:13 - main - INFO - h3: 10.0.0.3
2025-06-24 15:16:13 - main - INFO - h4: 10.0.0.4
2025-06-24 15:16:16 - src.topologies.rule_tester - INFO - Testing connectivity between all hosts with 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)
2025-06-24 15:16:16 - src.topologies.rule_tester - INFO - pingAll result: 0.0
  
```

Figura 3: Pingall

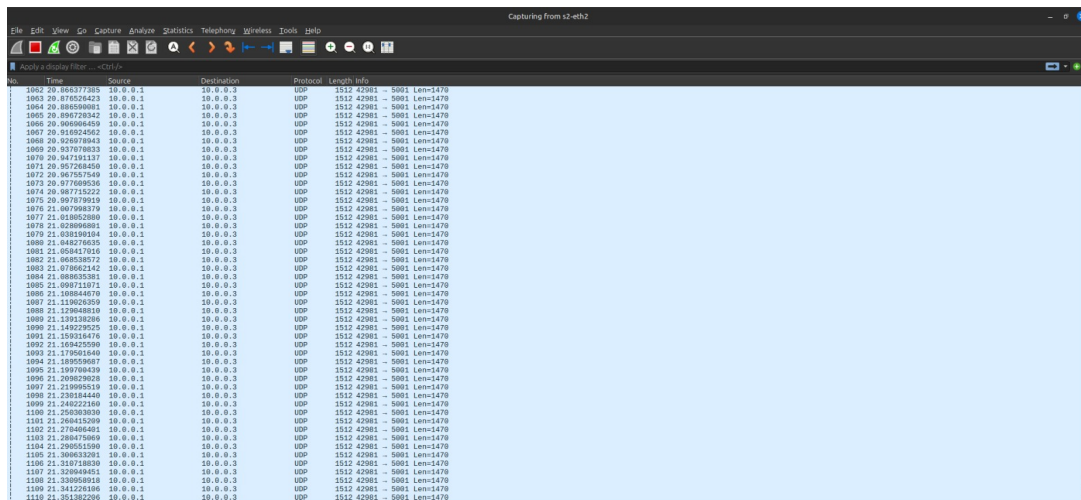
## 5.2. Pruebas de bloqueo con reglas de firewall

Posteriormente, se activaron las reglas definidas en el archivo `rules.json` y se repitieron las pruebas de conectividad, en este caso enfocadas en generar tráfico coincidente con las reglas de filtrado. Para cada una de las reglas, se tomaron capturas de paquetes con Wireshark y se observaron los mensajes de log emitidos por el controlador. A continuación se presentan los resultados correspondientes para la regla N2, debido que para el resto de reglas se cumple el mismo resultado, únicamente varía la regla ejecutada.

### ■ Regla 2: Bloqueo UDP desde IP específica al puerto 5001

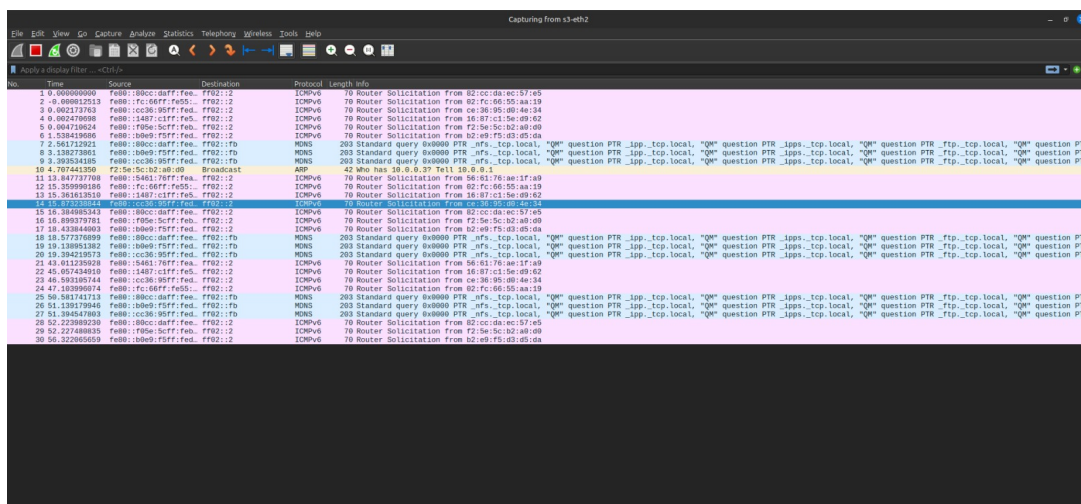
Se utilizó `iperf` para generar tráfico UDP desde el host 10.0.0.1 hacia el puerto 5001 del host 10.0.0.3. El tráfico fue correctamente filtrado, como lo demuestra la ausencia de paquetes en la captura y los mensajes de log que indican la coincidencia con la regla. En esta oportunidad, eligiendo "s3" como switch al cual se le instalan las reglas, podemos anticipar que los paquetes irán pasando a través de s1 y s2, y luego en s3 serán finalmente dropeados. Observemos en la captura siguiente como parándonos sobre s2-eth2 se observan paquetes, pero ya en s3-eth2 (siguiente captura) los paquetes son totalmente descartados por el switch debido a que los paquetes matchean con la regla definida en `rules.json`. De esta forma el Firewall está cumpliendo perfectamente su trabajo.





No.	Time	Source	Destination	Protocol	Length	Info
1802	20.866377303	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1803	20.876532423	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1804	20.886598081	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1805	20.896772042	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1806	20.906898459	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1807	20.916924662	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1808	20.926978843	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1809	20.937070633	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1870	20.947351137	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1871	20.957526450	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1872	20.967557549	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1873	20.977699536	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1874	20.987715222	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1875	20.997809219	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1876	21.007898279	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1877	21.018005200	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1878	21.028096881	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1879	21.038198584	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1880	21.048270335	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1881	21.058417516	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1882	21.068538072	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1883	21.078662162	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1884	21.088803281	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1885	21.098944870	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1886	21.109082359	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1887	21.119224819	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1888	21.129364810	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1889	21.139513286	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1890	21.149653225	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1891	21.159816476	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1892	21.169959299	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1893	21.179961840	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1894	21.189959807	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1895	21.199700439	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1896	21.209629820	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1897	21.219690519	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1898	21.230184440	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1899	21.240222480	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1900	21.250303030	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1901	21.260431099	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1902	21.270486481	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1903	21.280677809	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1904	21.290853180	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1905	21.300963291	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1906	21.310718030	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1907	21.320840451	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1908	21.330958818	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1909	21.341235186	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470
1910	21.351382206	10.0.0.1	10.0.0.3	UDP	1512	23901 → 5001 Len=1470

Figura 4: s2-eth2



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::b8cc:daff::fe::f802::2
2	0.000012513	fe80::fc:06ff::fe5::f802::2	fe80::fc:06ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::fc:06ff::fe5::f802::2
3	0.002173763	fe80::cc36:95ff::fe::f802::2	fe80::cc36:95ff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::cc36:95ff::fe::f802::2
4	0.002370908	fe80::1487:c1ff::fe5::f802::2	fe80::1487:c1ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::1487:c1ff::fe5::f802::2
5	0.004738024	fe80::f856:5c7f::fe5::f802::2	fe80::f856:5c7f::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::f856:5c7f::fe5::f802::2
6	1.53043966	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
7	2.56172921	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
8	3.138273981	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
9	3.393234185	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
10	4.707441350	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
11	11.847737708	fe80::1487:c1ff::fe5::f802::2	fe80::1487:c1ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::1487:c1ff::fe5::f802::2
12	15.359990186	fe80::fc:06ff::fe5::f802::2	fe80::fc:06ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::fc:06ff::fe5::f802::2
13	15.363813510	fe80::1487:c1ff::fe5::f802::2	fe80::1487:c1ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::1487:c1ff::fe5::f802::2
14	16.363813510	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::b8cc:daff::fe::f802::2
15	16.363813510	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::b8cc:daff::fe::f802::2
16	16.893379781	fe80::f856:5c7f::fe5::f802::2	fe80::f856:5c7f::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::f856:5c7f::fe5::f802::2
17	16.433848003	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::b8cc:daff::fe::f802::2
18	16.577370899	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
19	16.138051382	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
20	19.384219573	fe80::cc36:95ff::fe::f802::2	fe80::cc36:95ff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::cc36:95ff::fe::f802::2
21	43.612339318	fe80::1487:c1ff::fe5::f802::2	fe80::1487:c1ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::1487:c1ff::fe5::f802::2
22	45.657434919	fe80::1487:c1ff::fe5::f802::2	fe80::1487:c1ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::1487:c1ff::fe5::f802::2
23	46.593351144	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::b8cc:daff::fe::f802::2
24	47.103999074	fe80::fc:06ff::fe5::f802::2	fe80::fc:06ff::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::fc:06ff::fe5::f802::2
25	58.582741713	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
26	51.138179546	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	MNCS	203	Standard query fe80::b8cc:daff::fe::f802::2
27	51.384547603	fe80::cc36:95ff::fe::f802::2	fe80::cc36:95ff::fe::f802::2	MNCS	203	Standard query fe80::cc36:95ff::fe::f802::2
28	52.227889238	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::b8cc:daff::fe::f802::2
29	52.227889238	fe80::f856:5c7f::fe5::f802::2	fe80::f856:5c7f::fe5::f802::2	ICMPv6	70	Router Solicitation from fe80::f856:5c7f::fe5::f802::2
30	56.322889859	fe80::b8cc:daff::fe::f802::2	fe80::b8cc:daff::fe::f802::2	ICMPv6	70	Router Solicitation from fe80::b8cc:daff::fe::f802::2

Figura 5: s3-eth2

## 5.3. Logs del controlador

Durante las pruebas, el controlador registró eventos clave como el establecimiento de conexiones, la instalación de reglas de flujo y los paquetes descartados por coincidencia con reglas. Esto permitió una trazabilidad completa del comportamiento del firewall y facilitó la validación de su lógica.

En la siguiente captura se puede observar como el controlador:

- Informa en que switch instalara las reglas
- Informa que un switch se conecto a la red (en este caso el switch 2)
- Ignora al switch 2 y no le instala las reglas

```
INFO:custom.firewall:Firewall will be installed only on switch: s3
INFO:custom.switch_controller:Switch connected: [00-00-00-00-00-02 7]
INFO:custom.firewall:Switch s2 (dpid 2) is not the firewall switch (s3), skipping rule install.
```

Figura 6: Logs controlador

En la terminal del controlador se puede observar mucha más información, como por ejemplo: la correcta ejecución del **L2 learning**. Aquí vemos como el controlador:

- Aprendió correctamente a donde debe reenviar el paquete
- En caso de no saber a donde reenviar el paquete, ejecuta **FLOOD**.

```
INFO:custom.switch_controller:🔥 Forwarding packet by FLOOD (unknown MAC)
INFO:custom.switch_controller:🔥 Forwarding packet to port 3 (known MAC)
INFO:custom.switch_controller:🔥 Forwarding packet to port 1 (known MAC)
INFO:custom.switch_controller:🔥 Forwarding packet to port 3 (known MAC)
INFO:custom.switch_controller:🔥 Forwarding packet to port 3 (known MAC)
INFO:custom.switch_controller:🔥 Forwarding packet to port 3 (known MAC)
INFO:custom.switch_controller:🔥 Forwarding packet to port 2 (known MAC)
INFO:custom.switch_controller:🔥 Forwarding packet to port 1 (known MAC)
```

Figura 7: Logs controlador

## 6. Dificultades Encontradas

Durante el desarrollo del presente trabajo práctico se presentaron diversas dificultades técnicas y conceptuales que requirieron investigación exhaustiva y múltiples iteraciones para su resolución. A continuación se detallan los principales obstáculos encontrados:

### 6.1. Configuración Inicial del Entorno SDN

La primera barrera significativa fue establecer un entorno de desarrollo funcional que integrara correctamente Mininet, POX y OpenFlow. El proceso de configuración inicial presentó múltiples incompatibilidades entre versiones de Python, dependencias del sistema y bibliotecas requeridas.

Específicamente, POX requiere Python 2.7 para funcionar óptimamente, mientras que Mininet y las herramientas modernas están migrando hacia Python 3.x. Esta discrepancia generó conflictos en el entorno virtual y requirió la implementación de una solución híbrida que permitiera la coexistencia de ambas versiones de Python. La solución final involucró la creación de scripts automatizados (`setup_environment.sh`) que gestionan estas dependencias de manera transparente.

### 6.2. Comprensión del Paradigma SDN

La transición conceptual desde redes tradicionales hacia el paradigma Software-Defined Networks representó un desafío considerable. Inicialmente resultó complejo comprender la separación fundamental entre el plano de control y el plano de datos, así como el rol central del controlador en la toma de decisiones de reenvío.

La comprensión de cómo los switches OpenFlow delegan completamente la inteligencia al controlador externo, contrastando con el comportamiento autónomo de switches tradicionales, requirió un proceso de aprendizaje iterativo mediante experimentación práctica y análisis de logs detallados.

### 6.3. Gestión de Tablas de Flujo OpenFlow

La programación directa de tablas de flujo mediante mensajes `ofp_flow_mod` presentó una curva de aprendizaje pronunciada. Los aspectos más desafiantes incluyeron:

- **Prioridades de reglas:** Establecer jerarquías correctas para evitar que reglas de L2 learning interfieran con políticas de firewall
- **Campos de matching:** Configurar apropiadamente los campos `nw_src`, `nw_dst`, `tp_dst` y `nw_proto` para lograr filtrado granular
- **Acciones de flujo:** Implementar correctamente el comportamiento de DROP mediante listas de acciones vacías

## 7. Preguntas a Responder

1. **¿Cuál es la diferencia entre un Switch y un Router? ¿Que tienen en común?**

Un switch se usa para conectar equipos dentro de una misma red local, como pueden ser varias computadoras en una oficina. Su tarea es decidir a qué equipo debe enviar los datos, basándose en la dirección física (MAC) de cada uno. Funciona principalmente en la capa 2 del modelo TCP/IP.

En cambio, un router se utiliza para establecer comunicación entre diferentes redes, como por ejemplo entre una red local y la red de Internet. Su funcionamiento se basa en el uso de direcciones IP, y opera en la capa de red (capa 3) del modelo TCP/IP. Gracias a esto, es capaz de tomar decisiones de enrutamiento que permiten enviar datos a destinos que se encuentran fuera de la red local, atravesando múltiples redes intermedias si es necesario.

Ambos dispositivos ayudan a que los datos lleguen a su destino, pero lo hacen de formas distintas. También tienen en común que pueden enviar información entre distintos dispositivos y que muchas veces se combinan en equipos más avanzados que cumplen ambas funciones.

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

Un switch convencional utiliza protocolos y reglas establecidas por el fabricante para tomar decisiones de reenvío, principalmente basadas en la dirección MAC. Debido a su pre-configuración, suelen tener inconvenientes para escalar al no poder modificar el software cuando es requerido. Tanto las funcionalidades del plano de control como del plano de datos se realizan en el mismo dispositivo.

Un switch OpenFlow, en cambio, forma parte del paradigma de redes definidas por software (SDN). A través del protocolo OpenFlow, recibe instrucciones desde

un controlador centralizado, lo que permite modificar dinámicamente su comportamiento. Esto separa el plano de control del plano de datos, brindando una red más flexible, programable y escalable. Además, OpenFlow es un estándar abierto, lo que facilita la interoperabilidad entre equipos de diferentes marcas.

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

En escenarios complejos como el intercambio entre sistemas autónomos (interASes) en Internet, los routers tienen un rol fundamental. Estos dispositivos soportan protocolos como BGP, que permiten compartir rutas entre miles de redes distintas. Además, los routers deben mantener grandes tablas de enrutamiento y tomar decisiones complejas en tiempo real.

Si bien los switches OpenFlow pueden ejecutar algunas funciones de enrutamiento, su diseño está más orientado a entornos controlados, como redes de centros de datos o campus universitarios. Para reemplazar completamente a un router en Internet, un switch OpenFlow debería tener capacidades similares de procesamiento, soporte de protocolos complejos y alta confiabilidad, lo cual hoy no es práctico ni común.

Por lo tanto, no es viable reemplazar todos los routers de Internet por switches OpenFlow. Ambos cumplen funciones distintas y son complementarios, no intercambiables en todos los casos.

## 8. Conclusión

Este trabajo práctico permitió afianzar los conceptos centrales del paradigma de Redes Definidas por Software (SDN) y su implementación mediante OpenFlow. A través del desarrollo de una topología flexible en Mininet y un controlador externo en POX, se evidenció la separación entre los planos de control y de datos, y su utilidad para gestionar la red de forma centralizada.

La implementación de un firewall SDN destacó el potencial de esta arquitectura en términos de seguridad. Al definir reglas de bloqueo desde un archivo externo, se logró modificar el comportamiento de la red en tiempo real sin necesidad de intervenir físicamente en los dispositivos. Esto demuestra cómo SDN permite adaptar la red a distintos escenarios de forma ágil y programable.

El uso de herramientas como Wireshark e iperf permitió validar empíricamente la efectividad del sistema, confirmando que las reglas de filtrado fueron correctamente aplicadas por el switch designado.

En resumen, esta experiencia mostró cómo tecnologías como OpenFlow y POX permiten construir infraestructuras de red dinámicas, seguras y fácilmente administrables, sentando las bases para avanzar hacia soluciones más complejas en el ámbito de las redes programables.