



Trabajo Práctico N°2

Software-Defined Networks

Introducción a los Sistemas Distribuidos

(75.43 - TA048)

26/06/2025

Ascencio Felipe Santino 110675

Burgos Moreno Daniel 110486

García Pizales Ignacio 105043

Levi Dolores 105993

Orive María Sol 91351

Índice

1. Hipótesis y suposiciones realizadas	2
2. Herramientas utilizadas	3
2.1. Mininet	3
2.1.1. Características Principales	3
2.2. Pox	3
2.2.1. Características Principales	3
2.3. Wireshark	3
2.3.1. Características Principales	4
2.4. Iperf	4
2.5. Ping	4
3. Implementación	5
3.1. Reglas establecidas	5
3.2. Tutorial de funcionamiento	6
3.2.1. Requisitos	6
3.2.2. Comandos	6
3.3. Imágenes de la aplicación	7
4. Cuestionario teórico	10
4.1. ¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?	10
4.2. ¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?	10
4.3. ¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow?	10
5. Conclusiones	12
5.1. Dificultades encontradas	12

Introducción

En este trabajo práctico se propone implementar una red basada en *Software Defined Networking* utilizando el protocolo OpenFlow. El objetivo principal es desarrollar una topología en Mininet y programar un controlador con POX que permita definir políticas de reenvío de paquetes, incluyendo la implementación de un firewall simple.

Para lograrlo, se utilizó Mininet como entorno de simulación, POX como controlador, y herramientas como Wireshark e iperf para validar el comportamiento de la red. A lo largo del trabajo se buscó entender cómo interactúan el plano de datos y el plano de control en una red SDN, y cómo es posible modificar dinámicamente el comportamiento de los switches a partir de reglas definidas en software.

El informe incluye tanto la parte teórica como los resultados obtenidos durante las pruebas, junto con una explicación del código desarrollado para la simulación y el controlador.

1. Hipótesis y suposiciones realizadas

Durante el desarrollo del trabajo se realizaron algunas hipótesis y suposiciones con el fin de facilitar la implementación del firewall, el diseño de la topología y la validación de las reglas de bloqueo.

- Se asumió que la topología implementada sigue una **estructura lineal** (cadena de switches) con **cuatro hosts** distribuidos en los extremos: **h1**, **h2**, **h3** y **h4**. Esta disposición permitió realizar pruebas de comunicación simples entre extremos de la red.
- Se consideró que todos los hosts y switches tienen **nombres, direcciones MAC e IP estáticas** determinadas automáticamente por Mininet, lo que permitió establecer reglas de filtrado con valores fijos conocidos.
- Se asumió que el **switch intermedio (con dpid = 2)** es el encargado de aplicar las reglas de filtrado. Por este motivo, todas las reglas del archivo **reglas.json** apuntan exclusivamente a dicho switch.
- Se asumió que el tráfico a bloquear se encuentra dentro del rango de tráfico **IPv4**, por lo que todas las reglas del firewall se definieron con el campo **dl_type = 0x800**.
- Se consideró que las herramientas de prueba como **iperf** y **ping** eran suficientes para validar la efectividad de las reglas de firewall, y que cualquier comportamiento anómalo sería visible mediante **Wireshark** o los logs del controlador.
- Por simplicidad, se asumió que las pruebas eran bidireccionales, por lo que si se deseaba bloquear comunicación entre dos hosts, se agregaron reglas explícitas para ambos sentidos (MAC origen/-destino y viceversa).

2. Herramientas utilizadas

Se explicará a continuación que son, como funcionan y para que sirve cada una de las herramientas específicas utilizadas en el proyecto.

2.1. Mininet

Mininet es una herramienta de emulación de redes de código abierto que permite crear redes virtuales realistas en una sola máquina.

Mininet emula una red compuesta por hosts, switches, controladores y enlaces, todo ejecutándose como procesos o espacios de nombres (namespaces) en el sistema operativo. Esto permite realizar pruebas de arquitecturas y protocolos de red sin necesidad de hardware físico.

2.1.1. Características Principales

- Emula hosts, switches, enlaces y controladores en tiempo real.
- Utiliza Open vSwitch como switch virtual compatible con OpenFlow.
- Permite ejecutar programas reales de red (como ping, iperf, tcpdump) en nodos virtuales.
- Soporta scripting en Python para automatizar la creación de topologías complejas.
- Es liviano y se ejecuta completamente en entornos virtualizados (como una máquina virtual o contenedor).

2.2. Pox

POX es una plataforma de software escrita en Python diseñada para el desarrollo de controladores y aplicaciones de redes definidas por software (SDN), utilizando principalmente el protocolo OpenFlow.

Surgió como sucesor del controlador NOX y es ampliamente usado en entornos educativos e investigaciones gracias a su enfoque ligero y flexible.

2.2.1. Características Principales

- **Controlador y switch en Python:** Inicialmente concebido como controlador OpenFlow, POX también puede funcionar como switch OpenFlow, ofreciendo una plataforma homogénea para construir software de red.
- **Soporte para OpenFlow 1.0 y extensiones OVS/Nicira:** Maneja el protocolo OpenFlow versión 1.0 y cuenta con soporte para extensiones específicas de Open vSwitch.
- **Componentes modulares:** POX usa componentes en Python (como `forwarding.12_learning`, `log.level` o `discovery`) que se pueden combinar al iniciarlo para definir su comportamiento.
- **Integración con Mininet:** Integración con Mininet: Viene incluido en la máquina virtual de Mininet, lo que facilita su uso para emulación de redes y creación de prototipos.

2.3. Wireshark

Wireshark es una herramienta de análisis de protocolos de red (network protocol analyzer) de código abierto y gratuito. Permite capturar y examinar en tiempo real el tráfico de red que pasa por una interfaz de red.

Permite a los usuarios inspeccionar paquetes individuales, facilitando la identificación de problemas, análisis de seguridad y depuración de protocolos.

2.3.1. Características Principales

- Captura en tiempo real de paquetes en redes Ethernet, Wi-Fi, y otras.
- Soporta una amplia variedad de protocolos de red, desde los más comunes (TCP, UDP, HTTP) hasta protocolos especializados.
- Interfaz gráfica intuitiva con filtros avanzados para búsqueda y análisis.
- Posibilidad de guardar y exportar capturas para un análisis posterior.
- Herramientas de reconstrucción de sesiones y análisis de tráfico en profundidad.

2.4. Iperf

iperf es una herramienta de línea de comandos que permite realizar pruebas de velocidad entre un servidor y un cliente a través de una conexión de red. Te indica cuántos megabits por segundo (Mbps) pueden transferirse entre los dos nodos, lo que es clave para evaluar la calidad de una red.

2.5. Ping

ping es una herramienta de red utilizada para verificar la conectividad entre dos dispositivos en una red IP. Funciona enviando paquetes ICMP (Internet Control Message Protocol) de solicitud de eco al dispositivo destino y esperando respuestas de eco. Con esta información, ping mide el tiempo que tarda en recibirse la respuesta, lo que permite evaluar la latencia y detectar posibles pérdidas de paquetes. Ping es fundamental para diagnosticar problemas de red básicos y confirmar que un host está activo y accesible.

3. Implementación

La implementación del firewall se realizó utilizando el controlador **POX**, programado en Python, junto con una topología definida en **Mininet**. El controlador permite aplicar reglas de filtrado sobre el tráfico de red, tomando como base un archivo externo en formato **JSON** que contiene las condiciones a bloquear.

El controlador escucha los eventos de conexión de los switches (**ConnectionUp**) y, al recibir uno, aplica las reglas correspondientes al **dpid** del switch involucrado. Para cada regla, se construye un objeto **ofp_match()** con los campos necesarios (MACs, IPs, puertos, protocolos), y se envía un mensaje **ofp_flow_mod()** al switch para instalar la regla en su tabla de flujos.

El archivo de reglas utilizado es externo al controlador y permite modificar el comportamiento del firewall sin tener que cambiar el código fuente. A continuación en la siguiente sección se muestran algunas de las reglas implementadas

Estas reglas se aplican únicamente al switch con **dpid = 2**, especificado en cada regla mediante el campo **"switches"**. Esto permite extender el controlador fácilmente para trabajar con múltiples switches y topologías más complejas.

Finalmente, se utilizaron herramientas como **iperf** para verificar el bloqueo de tráfico en puertos específicos, y **Wireshark** para observar los paquetes filtrados en tiempo real, validando así el correcto funcionamiento del controlador desarrollado.

3.1. Reglas establecidas

Respetando el enunciado del trabajo práctico, se implementaron tres reglas para filtrar paquetes en el firewall, detalladas a continuación:

- Se deben descartar todos los mensajes cuyo puerto destino sea 80.
- Se deben descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y estén utilizando el protocolo UDP.
- Se deben elegir dos hosts cualesquiera y los mismos no deben poder comunicarse de ninguna forma (En nuestro caso elegimos el 'h1' y el 'h4').

```
1 [
2   {
3     "switches": [2],
4     "tipo_protocolo_transporte": "tcp",
5     "puerto_destino": 80
6   },
7   {
8     "switches": [2],
9     "tipo_protocolo_transporte": "udp",
10    "puerto_destino": 80
11  },
12  {
13    "switches": [2],
14    "tipo_protocolo_transporte": "udp",
15    "puerto_destino": 5001,
16    "mac_origen": "00:00:00:00:00:01"
17  },
18  {
19    "switches": [2],
20    "mac_origen": "00:00:00:00:00:01",
21    "mac_destino": "00:00:00:00:00:04"
22  },
23  {
24    "switches": [2],
25    "mac_origen": "00:00:00:00:00:04",
```

```
26     "mac_destino": "00:00:00:00:00:01"  
27   }  
28 ]
```

Listing 1: Archivo 'JSON' de configuración de reglas

3.2. Tutorial de funcionamiento

3.2.1. Requisitos

- Python 3.x instalado
- Herramientas mencionadas arriba.

3.2.2. Comandos

Lanzar mininet con una topología de cadena de n switches

```
1 ./topologia_cadena.sh $CANTIDAD_SWITCHES_DESEADOS
```

Lanzar el controlador 'POX'

```
1 ./controlador.sh
```

Comando para levantar los 4 hosts

```
1 xterm h1 h2 h3 h4
```

Comando para levantar el servidor en un 'PUERTO' usando 'TCP' en 10.0.0.0 (host 'X')

```
1 iperf -s -p $PUERTO
```

Comando para enviar paquetes desde el cliente al servidor indicado usando 'TCP'

```
1 iperf -c 10.0.0.<X> -p $PUERTO
```

Comando para levantar el servidor en un 'PUERTO' usando 'UDP' en 10.0.0.0 (host 'X')

```
1 iperf -s -p $PUERTO -u
```

Comando para enviar paquetes desde el cliente al servidor indicado usando 'UDP'

```
1 iperf -c 10.0.0.<x> -p $PUERTO -u
```

3.3. Imágenes de la aplicación

```
loli@loli-VirtualBox:~/Documents/refactor_redes-main$ ./controlador.sh
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
reglas.json
DEBUG:controller:Habilitando el Firewall.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.10.12/Feb 4 2025 14:57:36)
DEBUG:core:Platform is Linux-6.8.0-59-generic-x86_64-with-glibc2.35
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
DEBUG:openflow.of_01:[00-00-00-00-00-02 4] Got early port status message for port 65534
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-02 4]
DEBUG:controller:Reglas del Firewall instaladas en 00-00-00-00-00-02
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 2]
DEBUG:controller:Reglas del Firewall instaladas en 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-03 3]
DEBUG:controller:Reglas del Firewall instaladas en 00-00-00-00-00-03
```

Figura 1: Resultado de levantar el Firewall

```
loli@loli-VirtualBox:~/Documents/refactor_redes-main$ ./topologia_cadena.sh 3
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
kill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3
*** Adding links:
(h1, s1) (h2, s1) (h3, s3) (h4, s3) (s1, s2) (s2, s3)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 3 switches
s1 s2 s3 ...
*** Starting CLI:
mininet> 9
```

Figura 2: Resultado de levantar el Mininet


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

Figura 3: Resultado de hacer 'pingall' con la restricción de la primer regla

```

"Node: h1"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 81 -u
Server listening on UDP port 81
UDP buffer size: 208 KByte (default)
[ 1] local 10.0.0.1 port 81 connected with 10.0.0.3 port 51482
[ ID] Interval      Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[ 1] 0.0000-10.0013 sec  1.25 MBytes  1.05 Mbits/sec   0.026 ms  0/894 (0%)

"Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.1 -p 81 -u
Client connecting to 10.0.0.1, UDP port 81
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
[ 1] local 10.0.0.3 port 51482 connected with 10.0.0.1 port 81
[ ID] Interval      Transfer     Bandwidth
[ 1] 0.0000-10.0041 sec  1.25 MBytes  1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 1] Server Report:
[ ID] Interval      Transfer     Bandwidth       Jitter   Lost/Total Datagrams
[ 1] 0.0000-10.0013 sec  1.25 MBytes  1.05 Mbits/sec   0.026 ms  0/894 (0%)
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#

```

Figura 4: No aplica Regla 1

```

"Node: h1"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 80
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)

"Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.1 -p 80
Client connecting to 10.0.0.1, TCP port 80
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
TCP window size: 85.3 KByte (default)
[ 1] local 10.0.0.3 port 37320 connected with 10.0.0.1 port 80
[ ID] Interval      Transfer     Bandwidth
[ 1] 0.0000-10.0166 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@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#

```

Figura 5: Regla 2 1° ejemplo

```

"Node: h1"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 80 -u
Server listening on UDP port 80
UDP buffer size: 208 KByte (default)

"Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.1 -p 80 -u
Client connecting to 10.0.0.1, UDP port 80
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
[ 1] local 10.0.0.3 port 37320 connected with 10.0.0.1 port 80
[ ID] Interval      Transfer     Bandwidth
[ 1] 0.0000-10.0166 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@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#

```

Figura 6: Regla 2 2° ejemplo

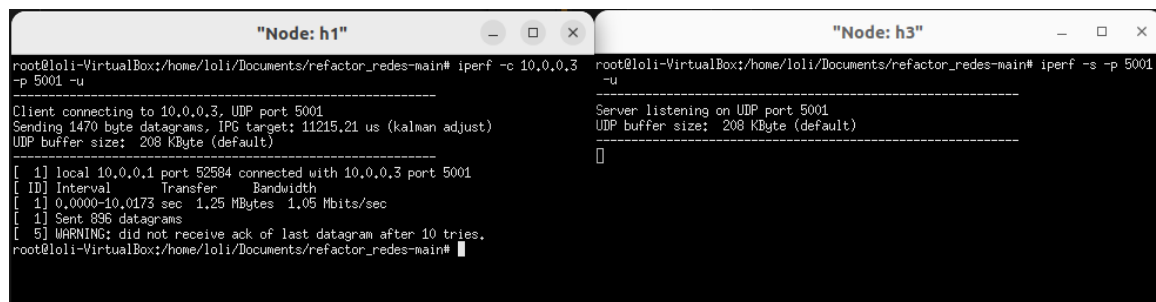
```

"Node: h1"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 81
Server listening on TCP port 81
TCP window size: 85.3 KByte (default)
[ 1] local 10.0.0.1 port 81 connected with 10.0.0.3 port 41366
[ ID] Interval      Transfer     Bandwidth
[ 1] 0.0000-3.9976 sec  27.3 GBytes  23.4 Gbits/sec

"Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.1 -p 81
Client connecting to 10.0.0.1, TCP port 81
TCP window size: 85.3 KByte (default)
[ 1] local 10.0.0.3 port 41366 connected with 10.0.0.1 port 81
[ ID] Interval      Transfer     Bandwidth
[ 1] 0.0000-10.0145 sec  27.3 GBytes  23.4 Gbits/sec
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#

```

Figura 7: No aplica Regla 2



```

"Node: h1"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.3 -p 5001 -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 52584 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0173 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@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#

"Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 5001 -u
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)

[]

```

Figura 8: No aplica Regla 3

4. Cuestionario teórico

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

Ambos son dispositivos de red que cumplen funciones diferentes, aunque ambos participan del encaminamiento de datos en una red. El **switch** conecta dispositivos dentro de una red, mientras que el **router** conecta diferentes redes entre sí.

El switch trabaja principalmente en la **capa de enlace de datos**. Su función principal es reenviar tramas entre dispositivos dentro de una red local (LAN), utilizando las direcciones *MAC* de origen y destino. Permite que dispositivos dentro de una misma red puedan comunicarse entre sí de manera eficiente.

Por otro lado, un router opera en la **capa de red** y su función es permitir la comunicación entre distintas redes. Los routers toman decisiones de encaminamiento en base a direcciones *IP*, utilizando tablas de ruteo y ejecutando protocolos como *OSPF*, *RIP* o *BGP*, entre otros.

En común, ambos:

- Participan del proceso de reenvío de paquetes.
- Disponen de una tabla interna que usan para tomar decisiones.
- Pueden ser configurados para aplicar políticas de red.
- Son componentes fundamentales de cualquier red.

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

La diferencia principal radica en el **nivel de control y flexibilidad** que se tiene sobre el comportamiento del dispositivo.

Un **switch convencional** tiene su lógica de reenvío fija o determinada por el fabricante. Aprende automáticamente las direcciones *MAC* que detecta por cada puerto y reenvía las tramas en base a esa información. Generalmente, las reglas de *forwarding* no pueden ser modificadas más allá de las opciones que brinda el sistema operativo o firmware del equipo.

En cambio, un **switch OpenFlow** está diseñado para funcionar dentro de una arquitectura **SDN**. Esto significa que **separa el plano de datos del plano de control**. A través del protocolo **OpenFlow**, un *controlador externo* tiene la capacidad de programar dinámicamente la *tabla de flujos* del switch. Cada entrada de esa tabla puede identificar un flujo determinado y especificar qué acción tomar con los paquetes que lo integran: reenviarlos, descartarlos, modificarlos, etc.

OpenFlow permite aplicar políticas complejas como filtrado por múltiples campos, *ingeniería de tráfico*, *NAT* o medidas de *seguridad* (como *firewalls*), todo de forma centralizada.

4.3. ¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow?

No, no es posible reemplazar todos los routers de la Internet por switches OpenFlow. Aunque la tecnología OpenFlow y las SDN ofrecen ventajas como mayor flexibilidad, capacidad de programación y control centralizado del tráfico, existen múltiples razones por las cuales no resultan adecuadas para operar en el plano global de Internet.

En primer lugar, la Internet está organizada como una red de redes, donde cada red de un proveedor o institución representa un **Sistema Autónomo (AS)** que toma decisiones de enrutamiento de manera independiente. Esta independencia se basa en protocolos como **BGP**, diseñados específicamente para

intercambiar información de ruteo entre ASes de forma descentralizada y resiliente. OpenFlow, en cambio, se apoya en un modelo centralizado donde los switches dependen de un *controlador externo* para recibir instrucciones, lo cual introduce un punto único de falla y limita la autonomía de cada red.

Además, escalar un controlador para administrar múltiples dominios independientes y redes con millones de rutas sería altamente complejo, tanto en términos de procesamiento como de latencia. Los routers actuales pueden tomar decisiones de reencaminamiento localmente y reaccionar rápidamente a fallos sin requerir coordinación externa. En cambio, un switch OpenFlow sin acceso constante a su controlador queda inoperante para flujos desconocidos.

También existen consideraciones prácticas: reemplazar la infraestructura actual implicaría una migración costosa y riesgosa. La mayoría de los routers que forman el backbone de Internet no soportan OpenFlow, y rediseñar toda la arquitectura para adaptarla a SDN requeriría un consenso global difícil de alcanzar.

5. Conclusiones

A lo largo de este trabajo práctico pudimos comprender mejor el funcionamiento de las **redes definidas por software** y el rol que cumple el protocolo **OpenFlow** en la separación del *plano de control* y el *plano de datos*. La posibilidad de programar el comportamiento de los switches desde un controlador externo representa un cambio importante respecto del modelo tradicional de redes, y abre la puerta a una mayor **flexibilidad y automatización**.

El uso de herramientas como **Mininet**, **POX** y **Wireshark** permitió no solo simular distintos escenarios, sino también observar cómo se aplican las reglas definidas y validar su correcto funcionamiento. La implementación del *firewall* nos dio una primera aproximación a cómo se pueden aplicar políticas de seguridad desde software, de forma dinámica y basada en flujos.

En resumen, el TP nos permitió **consolidar conceptos clave** de redes modernas, aprender nuevas herramientas y reflexionar sobre la evolución hacia infraestructuras más programables y adaptables.

5.1. Dificultades encontradas

Durante el desarrollo del trabajo práctico nos encontramos con varias dificultades técnicas y conceptuales que fuimos resolviendo a medida que avanzamos.

En primer lugar, **Mininet**, aunque es una herramienta muy útil para simular redes, presentó una curva de aprendizaje considerable. La lógica para construir topologías personalizadas, el uso de scripts en Python y la ejecución de comandos desde consola no siempre resultó intuitiva, sin embargo teníamos ya una experiencia adquirida después de la realización del primer trabajo práctico.

En cuanto al **controlador POX**, uno de los principales desafíos fue su instalación y configuración. Algunas distribuciones resultaban poco claras, con archivos dispersos y estructuras difíciles de interpretar. También nos llevó tiempo comprender cómo interactuar correctamente con la API, entender los eventos disponibles y cómo manipular paquetes desde el controlador.

El **pasaje de parámetros al configurar el firewall** fue otro punto que generó complicaciones. **La sintaxis requerida por POX** para establecer reglas específicas era muy sensible a detalles, lo cual llevó a errores difíciles de detectar al principio.

A pesar de estas dificultades, el proceso nos permitió adquirir una mejor comprensión de las herramientas utilizadas y valorar el nivel de control que ofrece SDN sobre el comportamiento de una red.