TP N°2: Software-Defined Networks

Redes (TA048)

Abril 2025

Mariano Gabriel Merlinsky Camins	110446
Facundo Andrés Calderan	110873
Matias Besmedrisnik	110487
Iván Alexis Maximoff	110868

Índice

1. Introducción

Este trabajo práctico tiene como objetivo acercarnos al funcionamiento de las SDN mediante la implementación de un firewall. La topología se emuló con mininet

2. Hipótesis y suposiciones realizadas

Algunas suposiciones que tomamos para realizar el trabajo son:

- Si no hay una regla definida para que tipo de protocolo se debe bloquear (UDP ó TCP), se bloque para ambos.
- Solamente se bloquea IPv4.
- Si no se especifica a que switch cargarle las reglas, se le cargan a todos los switches de la topología.

3. Implementación

3.1. POX

Si bien OpenFlow define el protocolo de comunicación entre switches y controlador, es necesario utilizar un software que implemente dicho protocolo y permita desarrollar aplicaciones que controlen la red. Para esto utilizaremos **POX**.

POX provee una API sencilla y eventos que permiten interactuar con los switches y aplicar reglas de red dinámicamente.

3.2. Controlador y reglas de firewall

Se utilizó el evento *_handle_ConnectionUp(self, event)* del controlador POX para instalar reglas específicas en un único switch designado como firewall. Esta función se ejecuta automáticamente cuando un switch se conecta al controlador.

Las reglas se almacenan en un archivo JSON llamado *firewall_policies.json*, el cual contiene una estructura que incluye:

- El *DPID* del switch al que se le deben aplicar las reglas de firewall.
- Una lista de reglas definidas por los siguientes campos:
  - *src_ip*: Dirección IP de origen.
  - *dst_ip*: Dirección IP de destino.
  - *src_port*: Puerto de origen.
  - *dst_port*: Puerto de destino.
  - *protocol*: Protocolo de transporte (TCP o UDP).

Si un paquete coincide con alguna de las reglas, este será descartado por el switch. De esta manera, se implementa un firewall flexible que puede ser reconfigurado simplemente editando el archivo `JSON`.

3.3. Topología

Se diseñó una topología parametrizable utilizando **Mininet**, con el objetivo de probar diferentes funcionalidades que brinda la tecnología **OpenFlow**. La topología consiste en una cadena lineal de switches, donde los extremos están conectados a dos hosts. La cantidad de switches es configurable por parámetro.

La siguiente figura muestra un ejemplo de la topología generada:

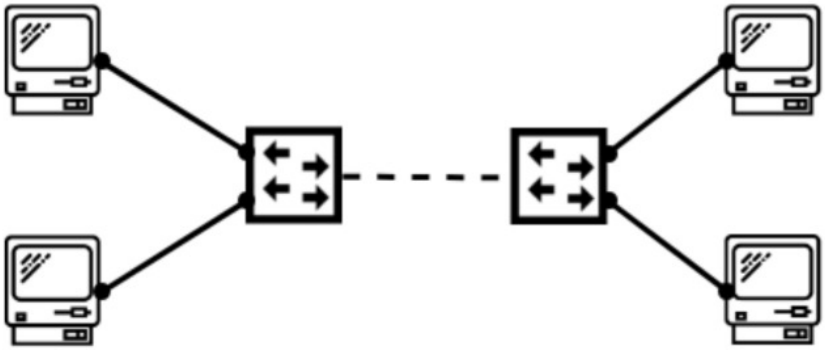

Figura 1: Topología en cadena con dos hosts y n switches

4. Pruebas

4.1. Objetivo

Evaluar el comportamiento de la red frente al bloqueo del puerto 80, comparando el tráfico TCP y UDP. Este caso permite observar cómo se manifiesta el bloqueo a nivel de cliente, servidor y en capturas realizadas con Wireshark.

4.2. Configuración Común

- **Regla aplicada**: Bloqueo de todo el tráfico dirigido al puerto 80.
- **Servidor**: h4 (IP: 10.0.0.4, puerto 80).
- **Cliente**: h1 (IP: 10.0.0.1).

4.3. Pruebas Automatizadas

El proyecto incluye un sistema completo de pruebas automatizadas documentado en el archivo `README.md`. Estas pruebas validan múltiples escenarios del firewall:

- **Demo automatizado**: Script `demo.sh` que ejecuta 4 casos de prueba completos
- **Bloqueos específicos**: Validación de reglas por puerto, protocolo e IP
- **Generación de logs**: Archivos de resultado organizados para análisis posterior
- **Capturas de red**: Archivos .pcap para análisis con Wireshark

Las pruebas presentadas a continuación se centran en el análisis detallado del comportamiento del firewall para el puerto 80, mostrando las diferencias entre protocolos TCP y UDP tanto a nivel de terminal como en capturas de red.

4.4. Escenario 1: Tráfico TCP

Comandos ejecutados

Servidor (h4):

```
# En terminal h4:
iperf -s -p 80
------------------------------------------------------------
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
# El servidor queda esperando conexiones que nunca llegan
```

Cliente (h1):

```
# En terminal h1:
iperf -c 10.0.0.4 -p 80 -t 10
tcp connect failed: Connection timed out
------------------------------------------------------------
Client connecting to 10.0.0.4, TCP port 80
TCP window size: -1.00 Byte (default)
------------------------------------------------------------
[  1] local 0.0.0.0 port 0 connected with 10.0.0.4 port 80
```

Captura en Wireshark (h1):

- **Filtro aplicado**: `tcp.port == 80`
- Resultado: retransmite debido a la falta de ACKs hasta que se produce el timeout.

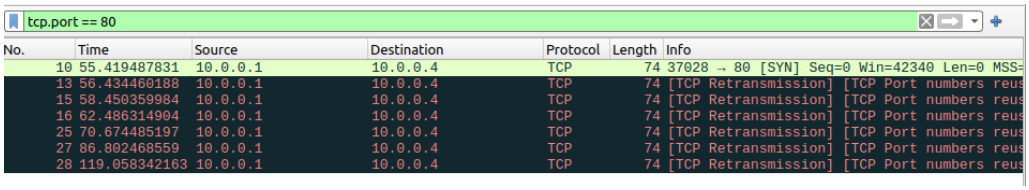

Figura 2: Captura de Wireshark - Tráfico TCP bloqueado

4.5. Escenario 2: Tráfico UDP

Comandos ejecutados

Servidor (h4):

```
# En terminal h4:
iperf -s -u -p 80
------------------------------------------------------------
Server listening on UDP port 80
Receiving 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
# El servidor permanece esperando datos que no llegan
```

Cliente (h1):

```
mininet> xterm h1
# En terminal h1:
iperf -c 10.0.0.4 -u -p 80 -t 5 -b 1M
------------------------------------------------------------
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 60121 connected with 10.0.0.4 port 80
[ ID] Interval       Transfer     Bandwidth
[  1] 0.0000-5.0134 sec   645 KBytes  1.05 Mbits/sec
[  1] Sent 450 datagrams
[  5] WARNING: did not receive ack of last datagram after 10 tries.
```

Captura en Wireshark (h1):

- **Filtro aplicado**: `udp.port == 80`
- Resultado: sólo tráfico saliente desde el cliente, sin respuestas del servidor.

udp.port == 80

No.	Time	Source	Destination	Protocol	Length	Info
102	790.390801269	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
103	790.402341898	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
104	790.402384706	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
105	790.413527631	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
106	790.424886567	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
107	790.435932170	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
108	790.447346356	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
109	790.458545351	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
110	790.469668038	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
111	790.480996900	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
112	790.492201371	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
113	790.503353018	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
114	790.514568964	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
115	790.525720028	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
116	790.537139147	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
117	790.548318917	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
118	790.559382534	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
119	790.570737991	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
120	790.581843228	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
121	790.593207397	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
122	790.604398707	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
123	790.615437738	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
124	790.626699148	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
125	790.638044273	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
126	790.649080490	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
127	790.660275626	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
128	790.671734515	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
129	790.682946560	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
130	790.694156954	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
131	790.705391289	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
132	790.716483561	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
133	790.727782051	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
134	790.738864520	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
135	790.750233986	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
136	790.761311157	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
137	790.772634017	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
138	790.783712026	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
139	790.795045573	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
140	790.806298227	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470
141	790.817553334	10.0.0.1	10.0.0.4	UDP	1512	60121 → 80 Len=1470

Figura 3: Captura de Wireshark - Tráfico UDP bloqueado

4.6. Comparación de Comportamientos

Aspecto	TCP Bloqueado	UDP Bloqueado
Síntomas en cliente	`Connection timed out`	`WARNING: no ACK final`
Síntomas en servidor	No se establece conexión	No se reciben datagramas
Wireshark	Solo tráfico saliente del cliente	Solo tráfico saliente del cliente

Estas pruebas nos permitieron entender las diferencias dependiendo del protocolo en el que se aplique el bloqueo:

- **TCP**: Dado que TCP es un protocolo orientado a conexión y confiable, cuando un paquete TCP es bloqueado por el firewall (por ejemplo, en un intento de conexión al puerto 80), se observa en Wireshark que el host emisor realiza intentos de retransmisión del paquete SYN.
- **UDP**: En este caso, dado que UDP no es confiable, simplemente se observa que los paquetes UDP salen del host emisor, pero no son respondidos ni reconocidos por el receptor. No hay retransmisiones, pero la falta de respuesta muestra que los paquetes fueron descartados (por ejemplo, si se bloqueó el puerto 80).

5. Preguntas a responder

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

   Los **switches** y **routers** son dispositivos esenciales para la comunicación en redes. Sin embargo, presentan algunas diferencias fundamentales:

- **Capas donde operan:** El switch funciona en la capa de enlace, y su función principal es permitir la comunicación entre dispositivos dentro de una misma red local (LAN). En cambio, el router opera en la capa de red y permite la comunicación entre distintas redes.
- **Envío de paquetes:** El switch reenvía paquetes utilizando las *direcciones MAC* y puerto de los dispositivos conectados. Además, su alcance está limitado a la red local. El router, por su parte, utiliza *direcciones IP* y protocolos de enrutamiento para tomar decisiones sobre el camino óptimo que debe seguir cada paquete, incluso a través de múltiples redes. También puede aplicar políticas de seguridad más complejas.

También comparten varias características ambos dispositivos permiten la conexión de equipos para facilitar la comunicación en red, toman decisiones sobre el manejo y reenvío de datos, pueden aplicar ciertas reglas de seguridad, y además pueden ser gestionados de forma centralizada como SDN.

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

   Su principal diferencia está en la forma en que toman decisiones de reenvío. Un switch convencional utiliza una lógica fija y se basa en el aprendizaje automático de direcciones MAC para decidir a qué puerto reenviar cada paquete.

   Por otro lado, los switches OpenFlow ofrecen mayor flexibilidad y control, ya que son gestionados externamente por un controlador centralizado que, utilizando el protocolo OpenFlow, toma decisiones acerca de cómo se deben procesar los paquetes. Esto permite aplicar reglas complejas que involucran capas superiores, como direcciones IP, protocolos y puertos. De esta manera, el switch puede comportarse como un firewall, realizar balanceo de carga o cumplir funciones de enrutamiento, según las instrucciones definidas por el controlador.

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

   A simple vista, no se podría reemplazar todos los routers por switches OpenFlow debido a la interacción entre sistemas autónomos (AS). La interacción entre sistemas autónomos utiliza mecanismos como BGP, el cual maneja atributos más complejos relacionados con relaciones comerciales. Este tipo de filtros no podría ser aplicable nunca en los switches OpenFlow.

   Además, este tipo de switches requiere de un controlador SDN centralizado, con la cantidad de rutas existentes, que un solo equipo maneje toda esa información en tiempo real es casi imposible y generaría un cuello de botella provocando poca tolerancia a fallos.

6. Dificultades encontradas

Al haber utilizado previamente Mininet para el trabajo práctico 1, la parte de armado de la topología nos resultó bastante llevadera. Las dificultades surgieron al tratar de comprender correctamente el funcionamiento de POX, ubicarse en la rama adecuada de su repositorio y manejar las versiones correspondientes de Python. En nuestro caso, utilizamos la versión 2.7.

7. NAT

Como bien sabemos, *Network Address Translation* (NAT) fue una solución al problema de la cantidad limitada de direcciones que presentaba IPv4.

En este trabajo práctico, de manera opcional, se propuso la realización de un NAT utilizando el controlador POX, con el objetivo de permitir que hosts en una red privada pudieran comunicarse con hosts dentro de una red pública mediante la traducción de direcciones.

Para esto, creamos una subred privada con los hosts *h1* y *h2*, donde implementamos el NAT en el switch más cercano a estos dos. La función *_handle_PacketIn* permite, de manera centralizada, que los paquetes enviados a ese switch pasen por el controlador para realizar la traducción NAT. Dentro de esta función, utilizamos una nat table que permite almacenar las relaciones NAT: *(ip,*

puerto) $\Rightarrow$ *(ip pública, puerto público).*

Podemos visualizar que al mandar paquetes desde `h1` a `h3`, se modifica la dirección IP de origen (`src_ip`) y el puerto de origen (`src_port`) del paquete:

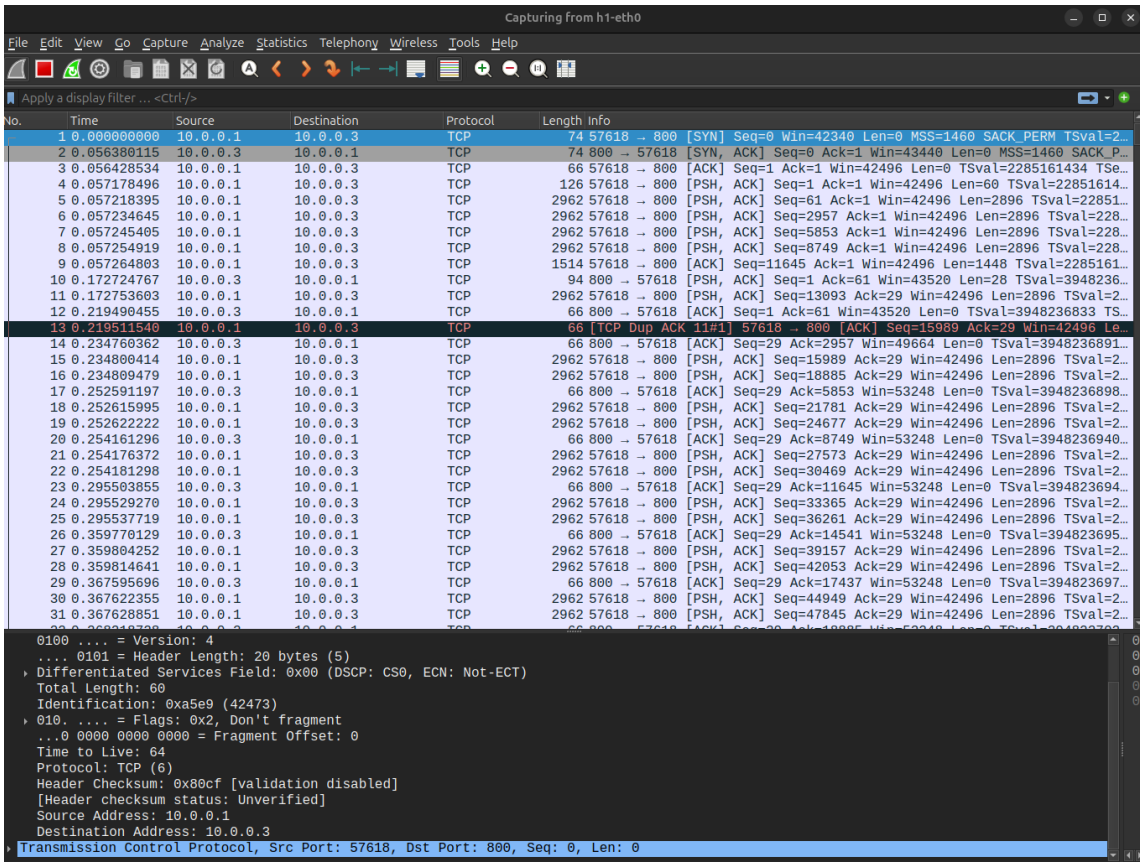

Figura 4: Vista de Wireshark desde `h1`

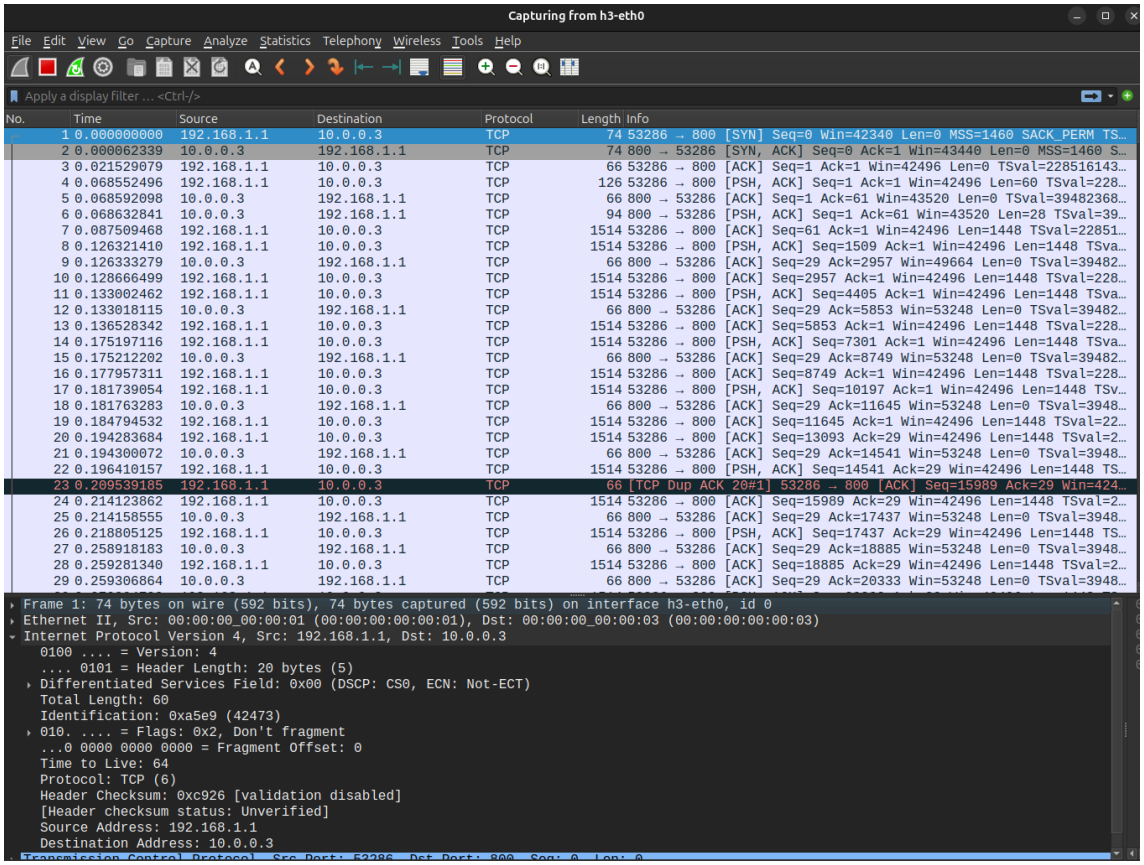

Figura 5: Vista de Wireshark desde `h3`

8. Conclusión

Este trabajo nos permitió comprender en mayor profundidad el funcionamiento de las *Software Defined Networks* (SDN), así como también la estructura y lógica detrás del protocolo *OpenFlow*. A través de la implementación de una topología personalizada y el desarrollo de un *firewall*, pudimos experimentar cómo se aplican reglas de red de forma dinámica y centralizada.

En resumen, el trabajo práctico nos brindó una visión práctica de cómo las redes pueden ser gestionadas mediante software, y cómo tecnologías como OpenFlow permiten aplicar políticas de seguridad y control.