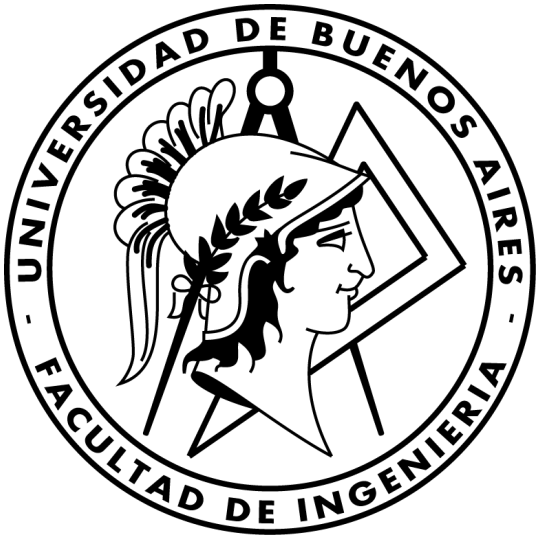


Universidad de Buenos Aires

Facultad de Ingeniería

Año 2023 - 2º Cuatrimestre

75.43 - Introducción a los Sistemas Distribuidos

TP 2
Software-Defined Networks

Grupo A3

Fecha de entrega: 21/11/2023

Nombre	Padrón	Correo
Longo Elia, Manuel	102425	mlongoe@fi.uba.ar
Romero, Adrián	103371	adromero@di.uba.ar
Craviotto Roitbarg, Mateo Exequiel	106255	macraviotto@fi.uba.ar
Roussilian, Juan Cruz	104269	jroussilian@fi.uba.ar
Prystupiuk, Maximiliano	94853	mprystupiuk@fi.uba.ar

Índice

1. Introducción

El objetivo de este trabajo es simular una SDN (Software-Defined Network) utilizando la herramienta Mininet, construyendo así una topología dinámica, y usando OpenFlow para poder implementar un Firewall a nivel de capa de enlace con reglas definidas en un archivo de configuración. La topología formada está compuesta por dos hosts conectados a un switch en un extremo, y otro par de hosts conectados a otro switch en el otro extremo. Los switches en los extremos estarán conectados a través de una cantidad variable de switches intermedios.

En cuanto al Firewall, se definen las siguientes reglas que debe cumplir:

1. Se deben descartar todos los mensajes cuyo puerto destino sea 80.
2. Se deben descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y estén utilizando el protocolo UDP.
3. Se debe elegir dos hosts cualquiera, y los mismos no deben poder comunicarse de ninguna forma

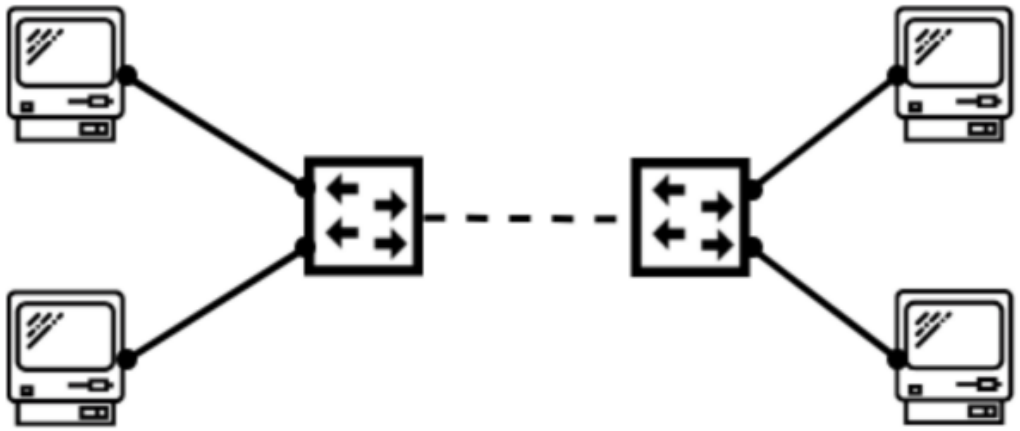
El funcionamiento de las reglas se comprueba mediante la herramienta *iperf y ping*.

2. Hipótesis y suposiciones realizadas

- El usuario debe pasar como parámetro la cantidad de switches intermedios a colocar en la topología
- La mínima cantidad de switches (total) en la topología es 1
- Los puertos utilizados por los servidores serán:
    - Servidor en el extremo izquierdo: Puerto 80
    - Servidor en el extremo derecho: Puerto 5001
- Se puede elegir cuáles de las reglas van a estar activadas mediante un archivo de configuración en formato json
- El firewall se instala en uno de los switches de los extremos únicamente como se mencionó en clase, por lo tanto las reglas del firewall no se aplican a los switches que están en el extremo contrario y se quieran comunicar entre sí

3. Implementación

Topología de la red



La topología de la red que creamos con Mininet está detallada en el archivo *Topology.py.*
El usuario invoca el programa con un parámetro para indicar la cantidad de switches totales que habrá en la topología. En caso de que no se indique la cantidad de switches, se utiliza el valor por defecto de este parámetro en 1, que es también el mínimo.

Para la creación de la topología se siguen los siguientes pasos:

- Se crean cuatro hosts (h1, h2, h3, h4)
- Se crea un switch inicial (s0)
- Se crean los N-1 switches restantes y se conectan los switches (s0 - s1 - … - sN)
- Se conectan h1 y h2 a al switch s0
- Se conectan h3 y h4 al switch sN

En cada uno de los extremos, uno de los hosts oficiara de servidor (específicamente h1 y h3)

Firewall y Reglas

Para el desarrollo del Firewall utilizamos la biblioteca de Pox e implementamos las reglas pedidas en el archivo *rules.json*, las cuales son:

1. Se bloquearán los paquetes que tengan como puerto destino el 80.
2. Se bloquearán los paquetes que provengan del host 1, tengan como puerto destino el 5001, y estén utilizando el protocolo UDP.
3. Se eligen dos hosts y no podrán comunicarse de ninguna forma.

El archivo de reglas se lee al iniciar el controlador y aplica aquellas reglas que tengan el campo *“enabled”* en *true*.

El formato del archivo *rules.json* es el siguiente:

- Campo *rules*: contiene la lista de reglas a aplicar
- Campo *firewall_switch:* indica el número de switch que actúa como firewall.

Dentro de cada regla se tienen los campos:

- *enabled*: Indica si la regla está activada (true) o no (false)
- *msg*: Mensaje que ve el usuario cuando la regla está activada
- rule: La regla en sí

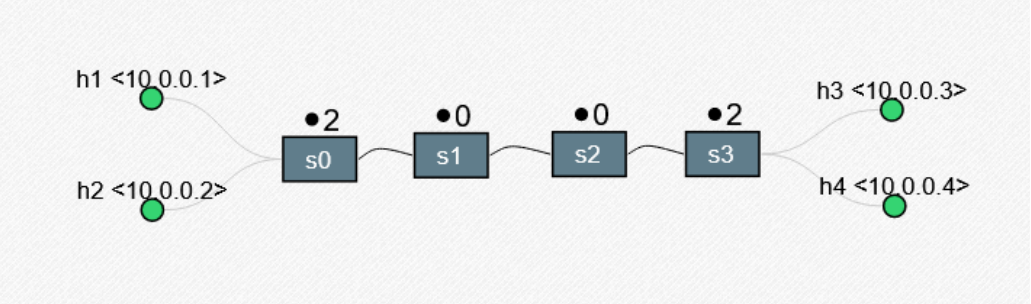
Luego dentro del campo *rule* se pueden tener los siguientes campos:

- *name*: Nombre de la regla
- *ip_type*: Versión del protocolo IP a utilizar (ipv4 o ipv6)
- *protocol:* Protocolo de capa de transporte a utilizar (udp o tcp)
- *dst_port:* Puerto de destino a bloquear
- *src_ip*: Dirección IP de origen a bloquear
- *src_mac*: Dirección MAC de origen a bloquear
- *dst_mac*: Dirección MAC de destino a bloquear

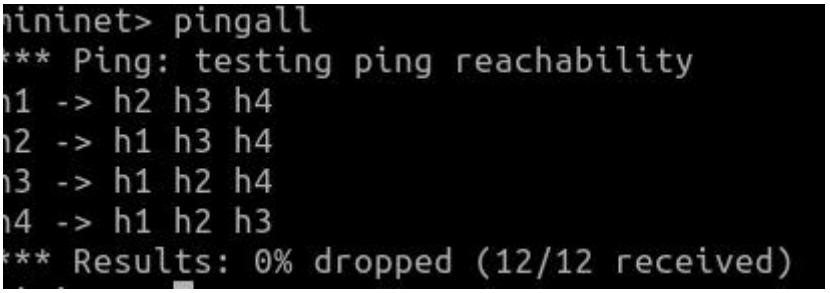
4. Pruebas

Pruebas de la topología

Verificamos que la topología sea la correcta graficando la misma mediante la herramienta SDN Spear de “Narmox”, la cual utiliza los comandos de mininet **dump** y **links**. Cabe destacar que en este caso la misma se levantó enviando como parámetro de cantidad de switches el nro. 4



También validamos que todos los nodos estén conectados ejecutando el comando *pingall*



Pruebas sobre el firewall

Pruebas sobre la regla 1 (bloquear paquetes puerto 80)

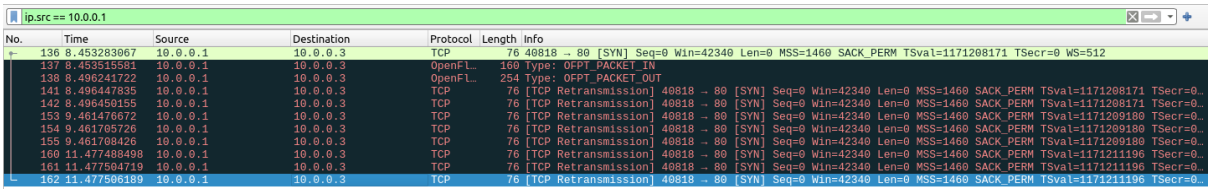
Para probar esta regla abrimos un servidor que acepta conexiones TCP en el host 3 que esté escuchando en el puerto 80.

```
root@manulon:/home/manulon/Escritorio/INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS/
TP2# iperf -s -p 80
------------------------------------------------------------
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

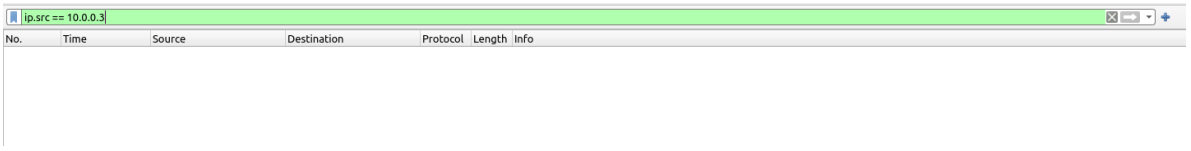
Luego abrimos un cliente en el host 1 que enviará paquetes TCP al servidor creado anteriormente.

```
root@manulon:/home/manulon/Escritorio/INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS/
TP2# iperf -c 10.0.0.3 -p 80
```

Si abrimos wireshark y vemos que lo está sucediendo en esta situación: el cliente envía el mensaje SYN para poder establecer una conexión TCP y como nunca recibe el SYN ACK este retransmite el paquete.



Para corroborar de que no se envía nada del lado del servidor, se corre el filtro de los paquetes que este envía. Y como se esperaba, el resultado da una lista vacía.

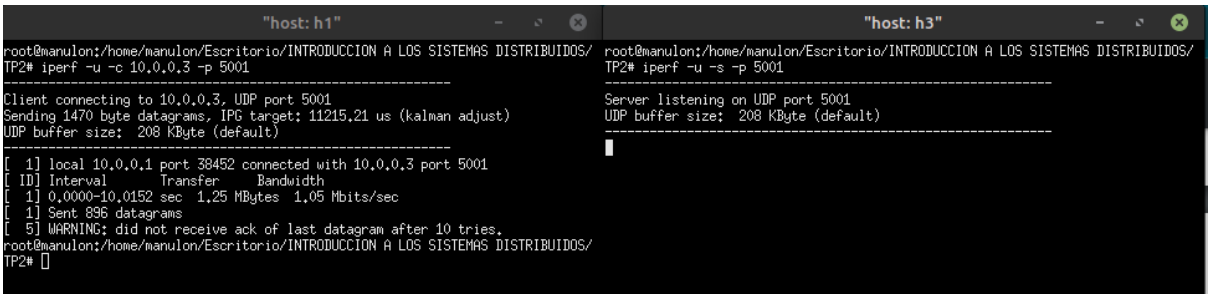


El controlador muestra que se bloquearán los paquetes que tengan como puerto destino el 80. Lo cual efectivamente fue el comportamiento encontrado.

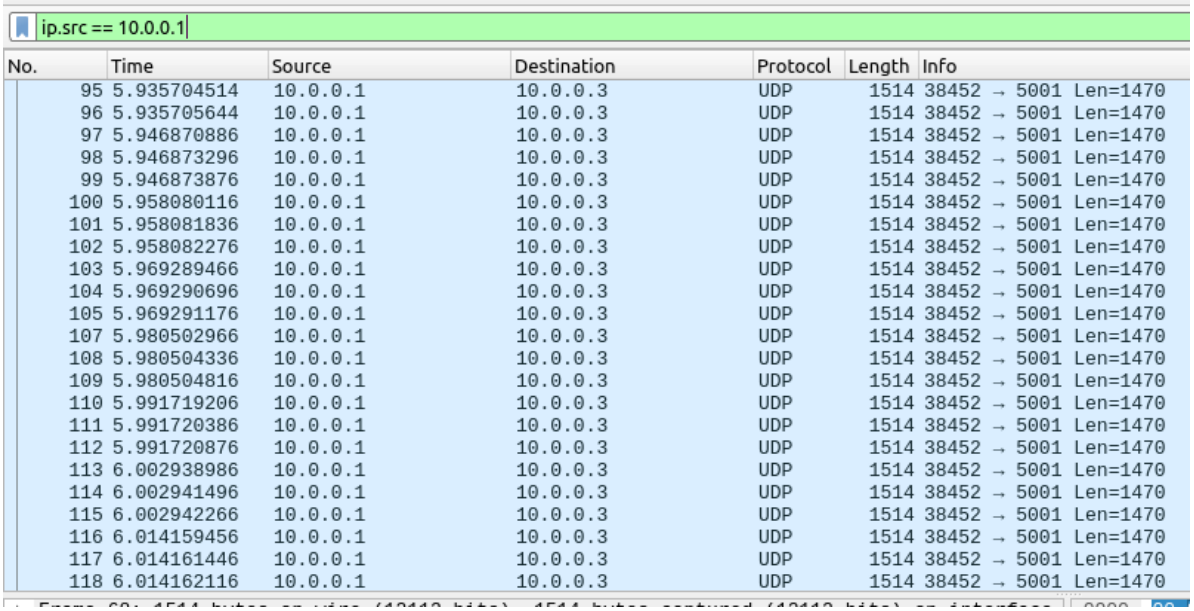
```
manulon@manulon:~/Escritorio/INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS/TP2$ pytho
n2 pox.py openflow.of_01 forwarding.l2_learning firewall
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[7a-fa-7a-d4-80-43 4] connected
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:firewall:Se bloquearan los paquetes que tengan como puerto destino el 80.
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
```

Pruebas sobre la regla 2 (bloquear paquetes que provengan del host 1, vayan al puerto 5001 y usen UDP)

Para poder comprobar que la regla, se levanta un servidor UDP que escucha en el puerto 5001 en un host diferente a h1 y se intenta enviar con la herramienta iperf paquetes udp. A continuación se ve una captura de el cliente h1 enviando una rafaga de paquetes UDP a h3 y en la consola de h3 no se ve la llegada de estos paquetes, aparte de mostrar que no se recibe el ACK del último datagrama enviado.

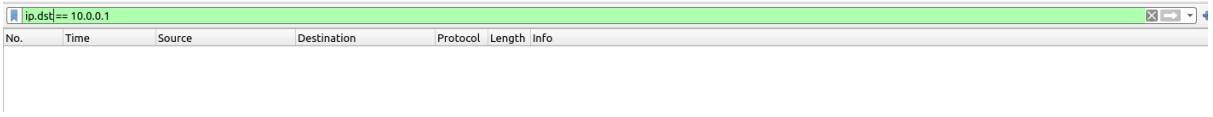


En wireshark se puede ver que se enviaron todos los paquetes



No.	Time	Source	Destination	Protocol	Length	Info
95	5.935704514	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
96	5.935705644	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
97	5.946870886	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
98	5.946873296	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
99	5.946873876	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
100	5.958080116	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
101	5.958081836	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
102	5.958082276	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
103	5.969289466	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
104	5.969290696	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
105	5.969291176	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
107	5.980502966	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
108	5.980504336	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
109	5.980504816	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
110	5.991719206	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
111	5.991720386	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
112	5.991720876	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
113	6.002938986	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
114	6.002941496	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
115	6.002942266	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
116	6.014159456	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
117	6.014161446	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470
118	6.014162116	10.0.0.1	10.0.0.3	UDP	1514	38452 → 5001 Len=1470

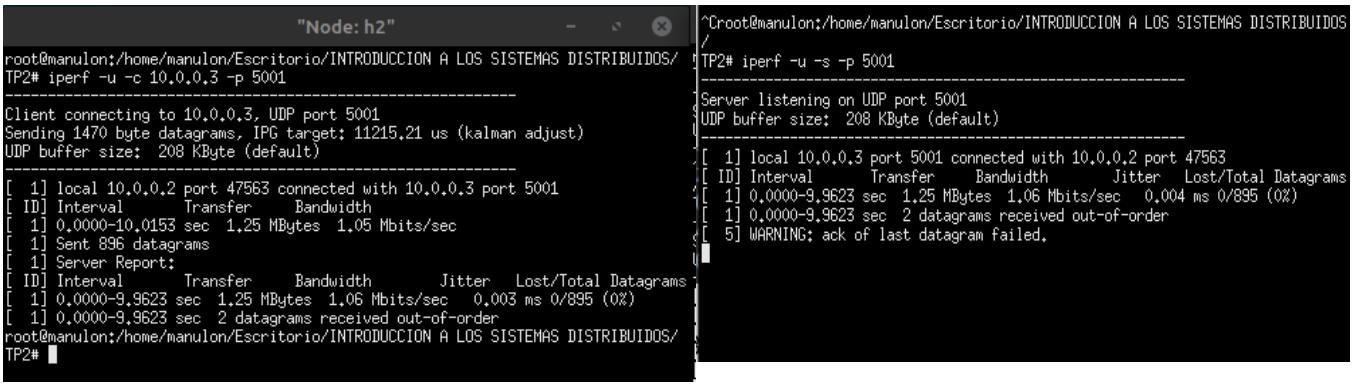
Pero el mismo host no recibe nada



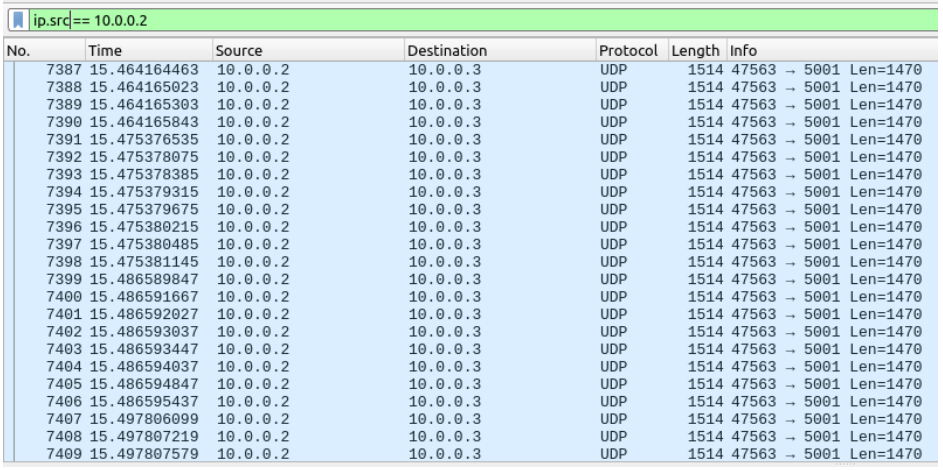
En la consola del firewall se puede ver como el controlador va a dropear todos los paquetes UDP provenientes de h1 con puerto destino 5001

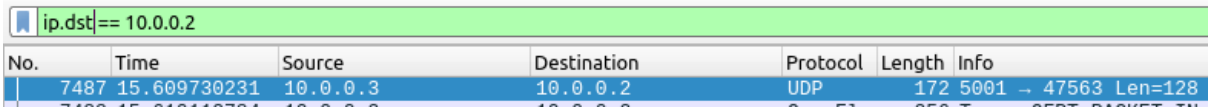
```
manulon@manulon:~/Escritorio/INTRODUCCION A LOS SISTEMAS DISTRIBUIDOS/TP2$ pytho
n2 pox.py openflow.of_01 forwarding.l2_learning firewall
POX 0.3.0 (dart) / Copyright 2011-2014 James McCauley, et al.
INFO:core:POX 0.3.0 (dart) is up.
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:firewall:Se bloquearan los paquetes que provengan del host 1, tengan como p
uerto destino el 5001, y esten utilizando el protocolo UDP.
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
INFO:openflow.of_01:[56-f3-c7-6b-9b-47 4] connected
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
```

En cambio, si se envía desde otro host (h3), este si se puede conectar y se puede ver en el servidor.



En wireshark se puede ver todos los paquetes que se envían y también se puede ver el último mensaje de ACK que envía el servidor al cliente.



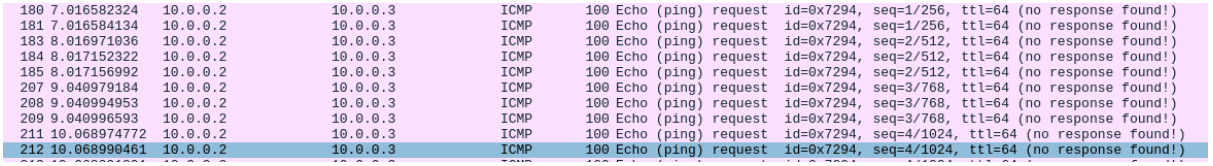


Pruebas sobre la regla 3 (dos host que no se puedan comunicar entre sí)

Para probar esto, primero decidimos probar hacer un ping desde h2 hacia h3 y luego al revés habiendo bloqueado la comunicación entre ambos hosts.
Los resultados que obtuvimos fueron los siguientes:

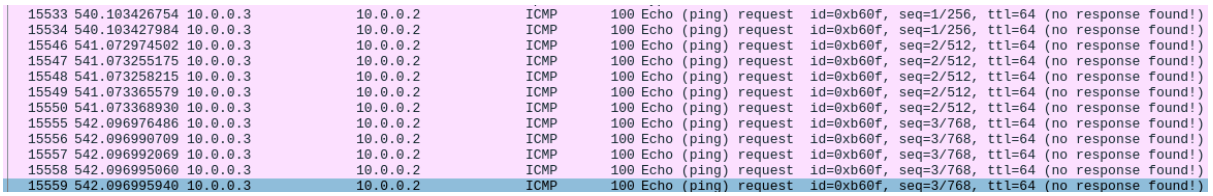
```
mininet> h2 ping h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
^C
--- 10.0.0.3 ping statistics ---
243 packets transmitted, 0 received, 100% packet loss, time 247795ms
```

Y en wireshark se pueden ver los paquetes salientes (ICMP echo) pero no los replies (echo reply).



Lo mismo pasa si hago el ping desde h3 hacia h2

```
mininet> h3 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
8 packets transmitted, 0 received, 100% packet loss, time 7166ms
```



También, al tirar pingall, los paquetes que no llegan son los que van de h1 a h2 o viceversa

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

5. Preguntas a responder

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

Un router opera en la capa de red, mientras que un Switch lo hace en la capa de enlace El router usa direcciones IP para saber a dónde enviar los paquetes, mientras que el Switch utiliza direcciones MAC. El Switch se suele usar para la interconexión de hosts dentro de una red local, y también se puede usar para interconectar routers. Por otra parte, el router se puede usar para interconectar hosts, y para conectarse a otras redes e interconectar redes locales.

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

Un Switch convencional únicamente almacena direcciones MAC para enviar paquetes. En cambio, el Switch OpenFlow controla varias tablas de flujo, donde es el propio servidor quien indica a los Switches el camino por el cual enviar los paquetes. Está diseñado para hacer pruebas de VLAN (Virtual Local Area Network), y para afrontar la movilidad de máquinas virtuales, redes móviles o redes con objetivos críticos.

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

No se pueden reemplazar todos los routers de Internet por Switches OpenFlow debido a varios inconvenientes. Primero, podría haber ataques del tipo Man in the Middle, ya que OpenFlow cuenta con su propio controlador centralizado. Además, podría generar problemas en el rendimiento ya que al utilizar OpenFlow, que usa reglas en lugar de IPs, se obtendrían tablas de ruteo enormes para el tráfico en Internet. El hecho de consultar esas tablas de gran tamaño resultaría en un peor rendimiento de la red.
En cuanto al escenario interASes, si se utilizaran Switches OpenFlow no se obtendría un resultado favorable, ya que el control centralizado de OpenFlow no escala bien para redes grandes como el caso del Internet. Además de esto, se necesitaría almacenar una cantidad enorme de entradas BGP, lo cual contribuye a que no sea viable implementarlo con Switches OpenFlow.

6. Dificultades encontradas

- La instalación de las herramientas Pox y OpenFlow fue complicada porque no había mucha documentación.
- Estuvimos obligados a utilizar dispositivos con sistema operativo Linux porque era más sencilla la instalación de las herramientas
- Nos costó comprender cómo levantar un servidor y un cliente en un nodo de la red de Mininet
- Nos tomó tiempo ver como pasar por parámetro la cantidad de switches de la topología por parámetro desde la terminal
- Contrastar con wireshark el funcionamiento de la topología

7. Conclusión

El trabajo práctico nos permitió construir una SDN definiendo la topología de la red con Mininet y usando OpenFlow para implementar un Firewall que filtre paquetes según diferentes reglas.

Además nos permitió comprender en profundidad qué es una SDN y cómo poder crear una con una topología específica, indicando la cantidad de switches, hosts y las conexiones entre los mismos.

Pudimos comprender cómo funciona el protocolo OpenFlow a través del cual se programan los dispositivos de red y que nos permite interactuar con las tablas de ruteo e implementar el Firewall.

Es vital para cuestiones de seguridad entender cómo funcionan los Firewalls y tener la capacidad de poder escribir uno por nuestra cuenta, lo cual hemos conseguido al finalizar esta entrega.