Universidad de Buenos Aires

Facultad de Ingeniería

Introducción a los Sistemas Distribuidos (75.43)

Primer cuatrimestre de 2024

TP N ∘ 2: SDN - Grupo 11

Lenguaje: Python

Chávez Cabanillas, José	96467	jchavez@fi.uba.ar
Davico, Mauricio	106171	mdavico@fi.uba.ar
Elias, Federico	96105	felias@fi.uba.ar
Hoszowski, Juan	104557	jhoszowski@fi.uba.ar
Hoszowski, Luciana	104554	lhoszowski@fi.uba.ar

Índice

1. Introducción

En este trabajo práctico se implementa un SDN (Firewall) con la ayuda de la librería POX. Para poder observar la funcionalidad del firewall se van a implementar unas simples reglas de bloqueo:

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 esten utilizando el protocolo UDP.
3. Se debe elegir dos hosts cualquiera, y los mismos no deben poder comunicarse de ninguna forma.

Estas reglas estaran en un archivo JSON modificable.

2. Implementación

2.1. Topología

Se realizó una topología parametrizable en la cual se tiene una cantidad variable de switches formando una cadena. Además en cada extremo se tienen dos hosts.

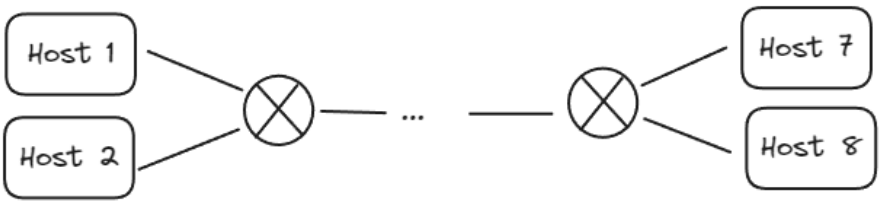

Figura 1: Topología usada

Para esto se utilizó Mininet, un emulador de red que permite crear y probar redes, y su API de Python: se crea una nueva clase llamada customTopo que hereda de la clase Topo. Esta clase representa la topología específica que se desea crear en Mininet. El constructor de la clase toma un parámetro llamado switches, que indica el número de switches que se desean en la topología. Luego, se crean 4 hosts (h1, h2, h7, h8) y se crean switches según el valor de switches. h1 y h2 se conectan al primer switch (s1). h7 y h8 se conectan al último switch (switches-1). Finalmente se establecen enlaces entre switches consecutivos (de s1 a switches-1).

2.2. Firewall

Para poder implementar este controlador/firewall que simulará un SDN de nuestra topología pedida debemos primero capturar todos los switches que se conecten a la red y agregarles a sus flow tables los match necesarios para bloquear los paquetes que entren en dicho switch. En la implementación pedida, solo capturamos la conexión de un switch específico y modificaremos su flow table.

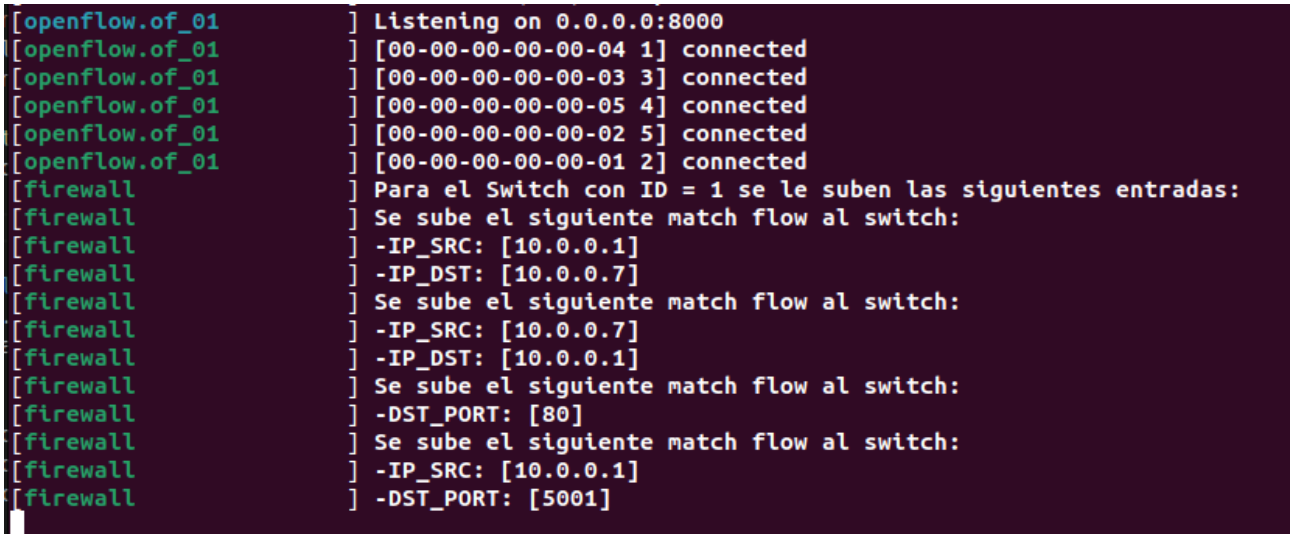

Figura 2: Captura de Switches en POX

Podemos ver en la figura 1 que POX detecta que entran 5 switches OpenFlow, y que a uno de ellos (elegido por linea de comando) se le agregan las entradas de su flow table para que funcione como firewall.

3. Ejecución

En un terminal ejecutar la topologia de mininet. Podemos elegir la cantidad de switches que queremos cambiando el `switches=5`, así como también elegir el puerto al cual pondremos nuestro controlador pox

```
sudo mn --custom firewall_topo.py --topo customTopo, switches=5
--mac --switch ovsk --controller=remote,port=6655
```

En otra terminal ejecutaremos nuestro firewall/SDN en el cual podremos elegir el switch que se utilizará como firewall con `--switch_id = n` y debemos utilizar el mismo puerto que utilizamos para mininet

```
python2.7 pox/pox.py log.level --INFO samples.pretty_log --openflow=INFO
openflow.of_01 --port=6655 forwarding.l2_learning firewall
--switch_id=1
```

En caso de error al ejecutar pox debido a un puerto ya utilizado, se debera cambiar a otro puerto en ambos mininet y pox (utilizar el mismo).

4. FireWall en acción

4.1. Regla 1

A continuación veremos cómo logramos que el switch elegido bloquee las comunicaciones pedidas utilizando iperf. Comprobaremos mediante iperf y wireshark la primer regla a seguir: que se descarten los mensajes con puerto destino 80. Para probarlo utilizamos iperf con el sigiente comando para el

host '10.0.0.1' que sera el cliente ejecuta: *iperf -c 10.0.0.8 -p 80* y para el host '10.0.0.8' que sera el servidor ejecuta : *iperf -s -p 80*

Source	Destination	Protocol	Length	Sequence Number	Bytes in flight	Info
10.0.0.1	10.0.0.8	TCP	74	0		33912 → 80 [SYN] Seq=0 Win=42340 Len=
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]
10.0.0.1	10.0.0.8	TCP	74	0		[TCP Retransmission] 33912 → 80 [SYN]

Figura 3: Captura de wireshark para la primer regla utilizando TCP

Podemos ver en la captura de la figura 2 como el cliente '10.0.0.1' intenta establecer una conexión TCP con el servidor '10.0.0.8' en el puerto 80 pero nunca recibe un SYN ACK por lo que podemos suponer que nuestro Switch está efectivamente dropeando los mensajes de '10.0.0.1'.

Probaremos ahora utilizando paquetes UDP con los siguientes comandos de iperf: host '10.0.0.1' que sera el cliente ejecuta : *iperf -c 10.0.0.8 -p 80 -u* y para el host '10.0.0.8' que será el servidor utilizamos el comando: *iperf -s -p 80 -u*

Source	Destination	Protocol	Length	Sequence Number	Bytes in flight	Info
fe80::200:ff:fe00:8	ff02::2	ICMPv6	70			Router Solicitation
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			44214 → 80 Len=1470

Figura 4: Captura de wireshark para la primer regla utilizando UDP

Podemos ver en la figura 3 como '10.0.0.1' le envía paquetes a '10.0.0.8' pero nunca recibe ACK.

4.2. Regla 2

Verificaremos que nuestro firewall cumple con la regla numero 2 probando primero utilizando paquetes TCP y luego paquetes UDP. Utilizaremos el comando *iperf -c 10.0.0.8 -p 5001* para el host '10.0.0.1' y el host '10.0.0.8' ejecutara: *iperf -s -p 5001* y obtendremos la siguiente captura:

Source	Destination	Protocol	Length	Sequence Number	Bytes in flight	Info
10.0.0.1	10.0.0.8	TCP	74	0		36678 → 5001 [SYN] Seq=
10.0.0.8	10.0.0.1	TCP	74	0		5001 → 36678 [SYN, ACK]
10.0.0.1	10.0.0.8	TCP	66	1		36678 → 5001 [ACK] Seq=
10.0.0.1	10.0.0.8	TCP	126	1	60	36678 → 5001 [PSH, ACK]
10.0.0.1	10.0.0.8	TCP	2962	61	2956	36678 → 5001 [PSH, ACK]
10.0.0.1	10.0.0.8	TCP	2962	2957	5852	36678 → 5001 [PSH, ACK]
10.0.0.1	10.0.0.8	TCP	2962	5853	8748	36678 → 5001 [PSH, ACK]
10.0.0.1	10.0.0.8	TCP	2962	8749	11644	36678 → 5001 [PSH, ACK]
10.0.0.1	10.0.0.8	TCP	1514	11645	13092	36678 → 5001 [ACK] Seq=
10.0.0.8	10.0.0.1	TCP	94	1	28	5001 → 36678 [PSH, ACK]

Figura 5: Captura de wireshark para la segunda regla utilizando TCP

Podemos observar en la figura 4 como el host '10.0.0.1' recibe un SYN ACK de '10.0.0.8' significando que nuestro switch no bloqueo la comunicación Pero si utilizamos UDP con los comandos *iperf -c 10.0.0.8 -p 5001 -u* y *iperf -s -p 5001 -u* obtenemos lo siguiente:

Source	Destination	Protocol	Length	Sequence Number	Bytes in flight	Info
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470
10.0.0.1	10.0.0.8	UDP	1512			51064 → 5001 Len=1470

Figura 6: Captura de wireshark para la segunda regla utilizando UDP

Observamos que '10.0.0.8' nunca le responde al host '10.0.0.1'.

4.3. Regla 3

Probaremos esta regla haremos que primero el host '10.0.0.1' intente comunicarse con '10.0.0.7' mediante TCP por un puerto y luego intentaremos que '10.0.0.7' sea el que inicie la conexión con otro puerto.

Source	Destination	Protocol	Length	Sequence Number	Bytes in flight	Info
10.0.0.1	10.0.0.7	TCP	74		0	36922 → 1234 [SYN] S
10.0.0.1	10.0.0.7	TCP	74		0	[TCP Retransmission]
10.0.0.1	10.0.0.7	TCP	74		0	[TCP Retransmission]
10.0.0.1	10.0.0.7	TCP	74		0	[TCP Retransmission]
10.0.0.1	10.0.0.7	TCP	74		0	[TCP Retransmission]
10.0.0.1	10.0.0.7	TCP	74		0	[TCP Retransmission]
10.0.0.1	10.0.0.7	TCP	74		0	[TCP Retransmission]

Figura 7: Captura de wireshark para la tercer regla de h1 a h7

Observamos que el switch bloquea los mensajes de '10.0.0.1' a '10.0.0.7'

Source	Destination	Protocol	Length	Sequence Number	Bytes in flight	Info
10.0.0.7	10.0.0.1	TCP	74		0	60156 → 7000 [SYN] S
10.0.0.7	10.0.0.1	TCP	74		0	[TCP Retransmission]
10.0.0.7	10.0.0.1	TCP	74		0	[TCP Retransmission]
10.0.0.7	10.0.0.1	TCP	74		0	[TCP Retransmission]
10.0.0.7	10.0.0.1	TCP	74		0	[TCP Retransmission]
fe80::4c64:eaff:fe5…	ff02::2	ICMPv6	70			Router Solicitation
10.0.0.7	10.0.0.1	TCP	74		0	[TCP Retransmission]

Figura 8: Captura de wireshark para la tercer regla de h7 a h1

En esta figura vemos que el switch bloquea la comunicación desde '10.0.0.7' a '10.0.0.1' correctamente

4.4. ¿Cuál es la diferencia entre un switch y un router? ¿Qué tienen en común?

Una característica de los switches que los routers no poseen es que son 'plug and play', es decir que para instalarse no necesitan ser configurados por un administrador ya que son 'self learning' y arman sus tablas en base a los frames que recibe. Otra importante diferencia es que los routers forwardean en base a la dirección IP mientras que los switches lo hacen en base a la dirección MAC. Una característica compartida es que aislan trafico y tiene ambos un control plane y un data plane.

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

En un switch convencional solo se aplican filtros basados en la MAC address destino, en cambio Los switches OpenFlow tambien tienen separado su control plane y su data plane permitiendo aplicar multiples reglas como por ejemplo el filtrado de IP's,de protocolo, etc... como tambien que se pueda configurar la tabla de flow de un switch. Esto permite un nivel de flexibilidad mayor.

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

Uno de los cons de utilizar switches es que funcionan mejor en redes pequeñas en parte por no poder configurar los caminos optimos de los paquetes a diferencia de los routers utilizan rutas mas inteligentes. Otro de los problemas de los switches es que no ofrecen el mismo nivel de control de tráfico que un router podría proporcionar. El hecho de que los switches OpenFlow pueden interactuar con multiples layers de comunicación podrían agregarle mucha mas complejidad a las redes de hoy en dia,

5. Conslusión

El trabajo presentado aborda la implementación y evaluación de un firewall utilizando SDN (Software-Defined Networking) en una topología de red simulada con Mininet. A lo largo del experimento, se verificaron tres reglas fundamentales del firewall mediante el uso de herramientas como iperf y Wireshark. Los resultados obtenidos muestran que el firewall fue capaz de bloquear o permitir tráfico específico según las reglas configuradas, demostrando así su efectividad en un entorno controlado.

La implementación del firewall se centró en la creación de reglas específicas para gestionar el tráfico TCP y UDP, demostrando cómo un controlador SDN puede ser programado para manipular las tablas de flujo de los switches OpenFlow y así cumplir con políticas de seguridad definidas. Además, se destacó la flexibilidad y programabilidad que ofrece SDN en comparación con las redes tradicionales, aunque también se señalaron desafíos como la necesidad de una infraestructura compatible y las preocupaciones de seguridad inherentes a la centralización del control.

En conclusión, la experiencia realizada con SDN y OpenFlow muestra que estas tecnologías tienen un gran potencial para mejorar la gestión y seguridad de las redes, ofreciendo una mayor capacidad de control y adaptación a las necesidades específicas de cada entorno. Sin embargo, es fundamental continuar investigando y realizando pruebas en escenarios más amplios y complejos para validar su eficacia y abordar las posibles limitaciones identificadas.