Universidad de Buenos Aires

Facultad de Ingeniería

2ndo Cuatrimestre de 2022

[75.43]

Introducción a los sistemas distruibuídos

Trabajo práctico 2: Software-Defined Networks

Integrantes: *Padrón:*

Fusco, Matias <mfusco@fi.uba.ar> 105683
Sardella, Florencia <fsardella@fi.uba.ar> 105717
Semorile, Gabriel <gsemorile@fi.uba.ar> 105681
Su, Agustina Doly <asu@fi.uba.ar> 105708
Szwarcberg, Tomás <tszwarcberg@fi.uba.ar> 103755

Links:

- Repositorio - GitHub.

Índice

1. Introducción

El presente trabajo práctico tiene como objetivo familiarizarse con los desafíos por los cuales surgen las SDNs y el protocolo OpenFlow, a través del cual se programan los dispositivos de red. Dado que ahora los dispositivos son programables, también se busca aprender a controlar el funcionamiento de los switches a través de una API.

Se plantea como objetivo construir una topología dinámica, donde se utilizará OpenFlow para poder implementar un Firewall a nivel de capa de enlace. Para poder plantear este escenario se emulará el comportamiento de la topología a través de mininet. El trabajo cuenta, con múltiples pasos y requisitos:

- **Base conceptual de la simulación**: Openflow (protocolo que permite que a través de un controlador central se definan políticas de como se deben enviar y clasificar los paquetes) y Firewall (componente de seguridad de red que monitorea el tráfico de red entrante y saliente y decide si permite el paso o bloquea un tráfico específico en función de un conjunto definido de reglas de seguridad).
- **Mininet**: Simulador de redes
- **Controladores de OpenFlow**: los controladores corren aplicaciones las cuales les permiten hacer uso del protocolo OpenFLow.

2. Hipótesis y suposiciones realizadas

A continuación se listan las hipótesis y suposiciones que se han tenido en cuenta para la realización de este trabajo práctico:

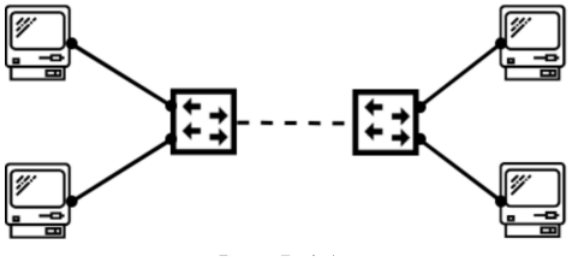
- El usuario usa la topología con mininet indicando la cantidad de switches a utilizar.
- Las reglas se activan y desactivan mediante un archivo json con las configuraciones.
- La configuración de hosts para las reglas se aplican desde un archivo json.
- El Firewall ignorará paquetes que no contengan IP como capa de red.
- Se puede seleccionar un switch como firewall desde la línea de comandos.
- Luego de iniciar el firewall no se requiere modificarlo.
- El switch indicado como firewall esta dentro de la topología (tiene un número de identificación menor o igual a la cantidad de switches de la topología).

3. Implementación

3.1. Topología

Para el presente trabajo práctico se desarrollo una topología parametrizable sobre la cual se prueban diferentes funcionalidades de OpenFlow.

Para la topología se tendrá una cantidad de switches variable, formando una cadena, en cuyos extremos se tienen dos hosts.



La topología debe recibir por parámetro la cantidad de switches, esto se hace mediante la consola de comandos:

```
florencia@HP-SAR:~$ sudo mn --custom /topology/LinearTopology.py  --topo linearTopo,2 --mac
--switch ovsk --controller remote
```

3.2. Firewall

Un switch dentro de la topología puede ser designado como Firewall, esto se indica mediante la línea de comandos:

```
florencia@HP-SAR:~/distribuidos/tp2_nuevo$ pox/pox.py openflow.of_01 forwarding.l2_learning
firewall.firewall --firewall_switch=1
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.1.0 (betta) is up.
```

Dicho Firewall se desarrolló utilizando la API de POX. POX es una plataforma para desarrollar controladores SDN con fines académicos y de investigación.

Se implementó para ello una clase Firewall que hereda de la clase de POX EventMixin que al recibir un evento de tipo ConnectionUp agrega las distintas reglas a la tabla de flujos. Dichas reglas no tienen acciones setteadas ya que justamente queremos que esos paquetes no se switcheen sino que finalicen ahí su transito por la red.

Las reglas implementadas son:

- **Regla 1:** descartar todos los mensajes cuyo puerto destino sea 80.
- **Regla 2:** descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y esten utilizando el protocolo UDP.
- **Regla 3:** seleccionar dos hosts cualquiera, y que los mismos no puedan comunicarse de ninguna forma.

Con el fin de conectar la red se utilizó l2_learning (herramienta provista por POX).

4. Pruebas

Para efectuar las pruebas y probar la implementación, lo primero que se hace es levantar POX con el Firewall creado y asignando un número de switch para que cumpla con esa funcionalidad:

```
root@1fdcbcb6947b:~# pox/pox.py openflow.of_01 forwarding.l2_learning firewall.firewall --fi
rewall_switch=1
POX 0.1.0 (betta) / Copyright 2011-2013 James McCauley, et al.
INFO:core:POX 0.1.0 (betta) is up.
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:firewall:Firewall set into switch: 1
INFO:firewall.rules.Rule1:Rule 1 applied
INFO:firewall.rules.Rule2:Rule 2 applied
INFO:firewall.rules.Rule3:Rule 3 applied
INFO:openflow.of_01:[00-00-00-00-00-02 3] connected
```

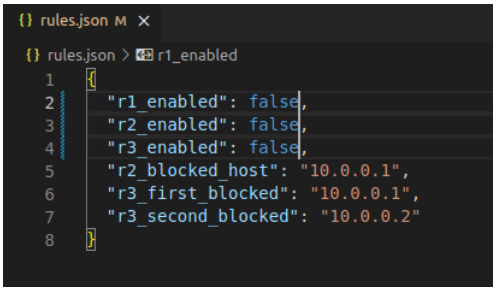
Luego, se levanta la topología utilizando mininet asignadole un controlador remoto. Este controlador remoto será el levantado con POX:

```
gabriel@gab-ub /mnt/882C57D72C57BEBE/gabriel/FIUBA/2C2022/Distribuidos/TPs/TP2/mininet-distribuidos
/topology  master ± sudo mn --custom LinearTopology.py  --topo linearTopo,3 --mac --switch ovsk -
-controller remote
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6633
*** 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>
```

Siendo que las reglas estan desactivadas y que se utiliza l2_learning para poder conectar la red, al hacer pingall en mininet todos los host deben poder comunicarse:

```
mininet> 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)
```

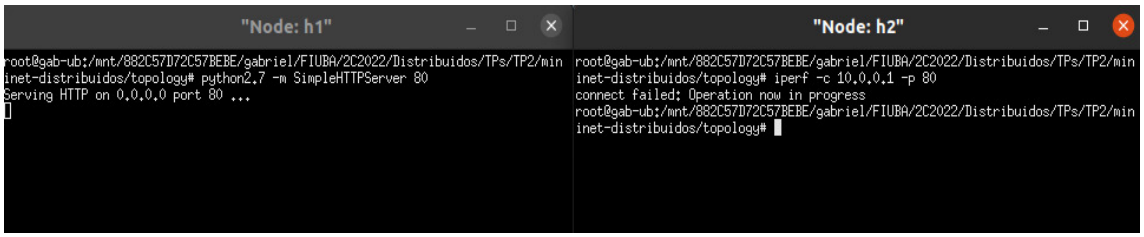
A continuación se probará por separado cada una de las reglas. La configuración de las mismas se encuentra en el archivo 'rules.json', en el mismo se pueden activar/-desactivar reglas así como también modificar los hosts implicados en ellas:



4.1. Regla 1

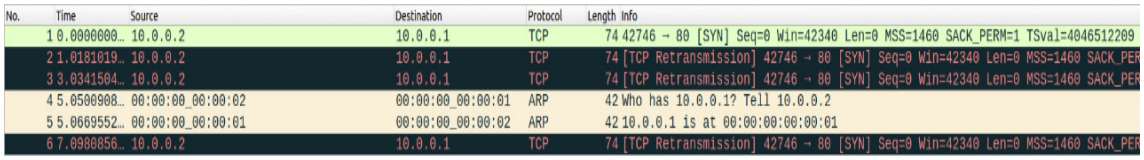
Se analiza ahora la primera regla, que indica que se deben descartar paquetes que sean destinados al puerto 80. Para ello se modifica el arhivo json para activar dicha regla.

Para ello se abrieron las consolas de 2 hosts y se asignó a uno como un servidor HTTP (puerto 80). Luego, utilizando la herramienta iperf que permite hacer pruebas en redes informáticas y cuyo funcionamiento habitual es crear flujos de datos TCP y UDP y medir el rendimiento de la red, se intentan enviar paquetes desde el segundo host al servidor HTTP:



Vemos como resultado en la consola del host 2 (cliente) que no se pudo establecer la conexión. Esto se debe a que se quiso establecer una conexión TCP y la misma requiere de un handshake previo en el que se intercambian 3 mensajes (SYN). Como el Firewall droppea los paquetes que se destinan al puerto 80 si la regla 1 esta activada (este caso) los mensajes destinados al handshake nunca llegan y por lo tanto la conexión no se puede establecer.

Esto mismo lo podemos ver en la corrida de wireshark:

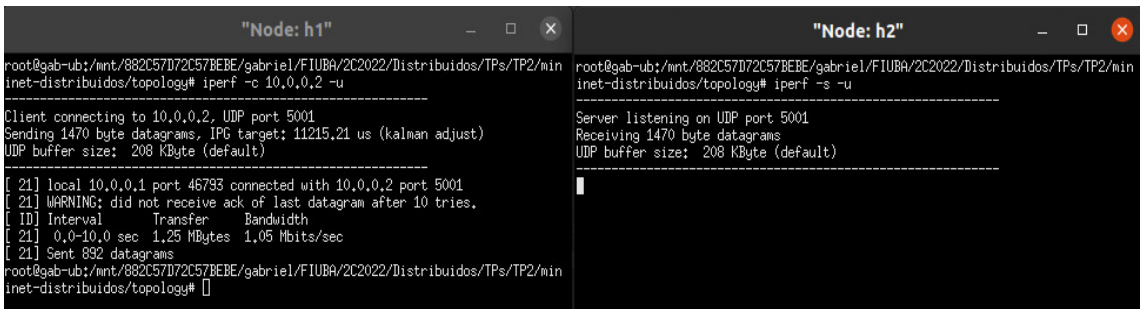


En la imagen podemos ver como el host 2 intenta mandarle un mensaje al host 1 y luego de varias retransmisiones establece que la conexión no se pudo establecer.

4.2. Regla 2

En este caso, se modifica el archivo json para que se active la regla 2. La regla 2 especifica que se deben descartar todos los paquetes que provengan del host 1, tengan puerto destino 5001 y utilicen protocolo UDP. Cabe aclarar que en caso de que se quiera intercambiar el host 1 por otro se puede configurar en el archivo json.

Para ello, nuevamente se abren las consolas del host 1 y del 2:



Como se puede ver, en la consola del host 1 se setea un cliente UDP y en la consola del host 2 se setea un servidor en el puerto 5001 que también utiliza UDP. Luego, al igual que con la regla anterior, se utiliza iperf para medir el rendimiento de querer enviar mensaje del host 1 al 2.

Como se observa, el host 2 nunca recibe un mensaje (notamos una diferencia con la regla 1 ya que en este caso se utiliza UDP y en el mismo no hay handshake). En la consola del host 1 vemos una advertencia que indica que no se recibió el ack de ninguno de los mensajes envíados y que se reintentó 10 veces.

Podemos observar esto mismo en la corrida del wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
875	9.7798155…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
876	9.7909492…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
877	9.8021127…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
878	9.8134202…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
879	9.8246233…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
880	9.8358517…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
881	9.8470511…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
882	9.8582422…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
883	9.8694531…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
884	9.8807001…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
885	9.8918647…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
886	9.9031459…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
887	9.9143407…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
888	9.9255432…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
889	9.9367848…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
890	9.9479497…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
891	9.9591465…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
892	9.9703346…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
893	9.9815686…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
894	9.9927887…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
895	10.003997…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
896	10.254114…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
897	10.504423…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
898	10.754710…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
899	11.005052…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
900	11.255439…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
901	11.505837…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
902	11.756159…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
903	12.006229…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470
904	12.256558…	10.0.0.1	10.0.0.2	UDP	1512	38042 → 5001 Len=1470

En la imagen podemos ver como hay varios envíos desde el host 1 al host 2 con protocolo UDP y como no hay niguna respuesta. Esto se debe a que el Firewall esta droppeando estos paquetes para cumplir con la regla.

4.3. Regla 3

Para este caso se activa en el json la regla 3. La misma específica que se deben seleccionar dos hosts cualesquiera y que ambos no deben poder comunicarse. Los hosts que no se pueden comunicar son configurados en el archivo json (en este caso se utiliza el 1 y el 2).

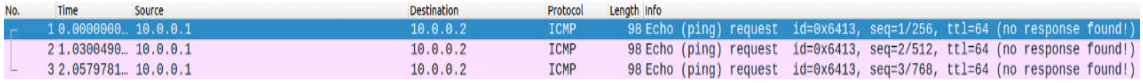
Para probar esta regla se prueba que el host 1 le pueda enviar un paquete al host 2 y viceversa.

Primero, se prueba que el host 1 no le pueda enviar un paquete al host 2:

```
mininet> h1 ping h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2058ms
```

Como ping no termina de reintentar, se lo corta manualmente utilizando ctrl+c. Vemos que en el transcurso de tiempo en el cuál corrió se intentaron enviar 3 mensajes y que ninguno llegó a destino.

Esto también lo podemos observar en wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000…	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x6413, seq=1/256, ttl=64 (no response found!)
2	1.0300490…	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x6413, seq=2/512, ttl=64 (no response found!)
3	2.0579781…	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x6413, seq=3/768, ttl=64 (no response found!)

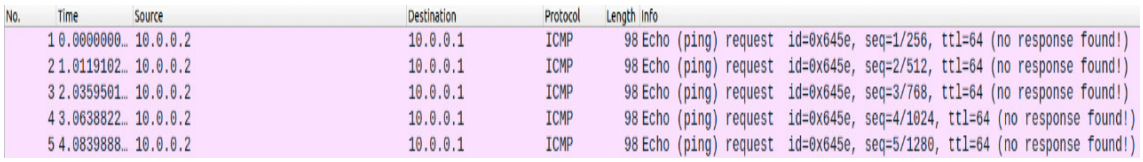
Como se puede ver en la imagen se intentaron enviar 3 mensajes del host 1 al host 2 y ninguno de ellos obtuvo respuesta.

Luego, se probó el camino opuesto, es decir, que el host 2 no le pueda enviar paquetes al host 1:

```
mininet> h2 ping h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
^C
--- 10.0.0.1 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4084ms
```

Se puede ver que al cortar la ejecución se llegaron a intentar enviar 5 paquetes pero niguno de ellos fue recibido por el host destino.

Nuevamente, se puede contemplar en la corrida de wireshark:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000…	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x645e, seq=1/256, ttl=64 (no response found!)
2	1.0119102…	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x645e, seq=2/512, ttl=64 (no response found!)
3	2.0359501…	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x645e, seq=3/768, ttl=64 (no response found!)
4	3.0638822…	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x645e, seq=4/1024, ttl=64 (no response found!)
5	4.0839888…	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x645e, seq=5/1280, ttl=64 (no response found!)

Se observa el envío de 5 mensajes del host 2 al host 1 y que niguno de ellos obtuvo una respuesta.

Finalmente, se probó hacer un pingall para comprobar que los únicos incomunicados debían ser el host 1 con el 2 y el 2 con el 1, el restulado de dicha operación fue el esperado y se puede ver a continuación:

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

5. Preguntas a responder

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

La principal diferencia entre un switch y un router es que el primero funciona sobre la capa de enlace mientras que el router en la de red.

La principal característica que comparten es el switchear paquetes. Sin embargo, el switch forwardea paquetes a partir de direcciones MAC, mientras que el router lo hace con las direcciones IP.

Además, el switch se encarga de crear una red conectando diferentes dispositivos entre ellos y el router establece la conexión entre estas redes.

Los routers se encargan de determinar la ruta mejor para que un paquete llegue a destino además de que los modifican (ej, TTL). En cambio, los switches reciben un paquete, lo procesan para determinar su dirección de destino y lo reenvián a esa dirección de destino.

Ambos dispositivos comparten la funcionalidad de proporcionar conectividad entre dispositivos, determinar la salida de un paquete a partir de las direcciones del mismo, y tener su arquitectura dividida en el plano de control y el plano de datos.

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

En un switch convencional, el reenvío de paquetes (data plane) y el encaminamiento de alto nivel (control plane) se realizan en el mismo dispositivo, sin embargo en los switches OpenFlow ambos se separan, el data plane se encuentra en el switch y para el control plane el switch se comunica con un controlador mediante el protocolo OpenFlow.

Un switch OpenFLow utiliza una tabla de flujos que permite generar políticas y comportamientos en base a flujos, además se comunica mediante una conexión encriptada con un controlador que configura la tabla de flujos.

Un switch convencional utiliza una tabla de Content Addressable Memory que determina el puerto de salida en función de la dirección MAC de destino.

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

No es posible reemplazar todos los routers de Internet por switches OpenFlow ya que, considerando el escenario interASes, los dispositivos que pertenecen al borde de la red deben implementar BGP para poder comunicarse con otros AS. Por otro lado, no a todas las redes les es útil el uso de las funcionalidades que provee openflow, por lo cual sería un sin sentido reemplazarlos dado que las tecnologías SDN poseen la desventaja de tener problemas de escalabilidad.

6. Dificultades encontradas

- Descargar POX.
- Descargar mininet en distintos sistemas operativos.
- Usar docker para poder hacer que a todos nos funcione igual POX y mininet.
- Conectarse con un container de docker corriendo para poder usar POX y mininet al mismo tiempo.
- Falta de documentación.

7. Conclusión

Tras haber implementado una solución para el trabajo práctico se pudo emular el funcionamiento de una red compuesta por una cantidad de switches variables y analizar su funcionamiento. Además, con la creación de un Firewall implementado en un switch específico pudimos comprender el funcionamiento del protocolo OpenFlow y lo que el mismo proporciona.

Finalmente, con lo nombrado anteriormente se pudo comprender en mayor profundidad, analizar y poner en práctica los conceptos teóricos sobre SDN aprendidos en clase. Además, pudimos seguir profundizando nuestros conocimientos de distintas herramientas como wireshark y aprender sobre algunas nuevas así como iperf, mininet y POX.