FACULTAD DE INGENIERIA
Universidad de Buenos Aires

TP2: Software-Defined Networks

[75.43] Redes
Segundo cuatrimestre del año 2024

Integrantes

Alumno	Padrón
Galián, Tomás Ezequiel	104354
Saez, Edgardo Francisco	104896
Pujato, Iñaki	109131
Lardiez, Mateo	107992
Zacarías, Víctor	107080

Índice

1. Introducción

Las redes definidas por software (Software-Defined Networks, SDN) son aquellas en las que los switches y routers que conforman una subred son gestionados de forma centralizada por el administrador (mediante un controlador remoto) de la subred vía software. Esto quiere decir que el admnistrador de la subred puede configurar y actualizar el funcionamiento de los switches y routers de la subred, así como definir políticas de filtrado de paquetes (firewall).

Una SDN no hace más que separar el plano de control del plano de los datos de la capa de red, garantizando que la gestión del plano de control sea una tarea de alto nivel, y que se pueda responder con mayor eficiencia a las requerimientos crecientes de la red.

A su vez, al hacer uso de estas prácticas, es necesario realizar simulaciones para validar que las configuraciones realizadas sobre los elementos de una subred tengan los resultados esperados.

Por ello, es de interés aplicar los fundamentos de las SDNs en la simulación de un firewall sobre una subred gestionada de forma centralizada. El objetivo no es más que el de validar hipótesis realizadas sobre el funcionamiento deseado de la red y mostrar la utilidad de las simulaciones en la gestión eficientes de las redes.

2. Herramientas de trabajo

- Mininet: Como programa de simulación de topologías de red, y de los elementos que la integran.
- POX: API de OpenFlow para Python.
- Iperf: Herramienta para levantar clientes y servidores de prueba.

3. Marco de trabajo

Para realizar la simulación fue necesario realizar un mix de las herramientas de trabajo. En primer lugar, se usó **Mininet** para simular una subred gestionada por un controlador remoto. Luego, para la simulación del controlador remoto se hizo un programa pequeño en Python que hacía uso de la API **POX** (el controlador remoto del programa en Python es quien gestiona a la red simulada con mininet). Por último, se hizo uso de **Iperf** para levantar los clientes y servidores sobre la red simulada y hacer que se envíen mensajes entre sí.

La red simulada no tiene una topología cualquiera sino que la misma sigue una arquitectura de N switches (es una topología parametrizable) conectados linealmente con dos hosts en ambos extremos de la cadena (4 hosts en total).

El controlador remoto programado simula a un firewall, y gestiona únicamente a los switches que integran la subred simulada con mininet (no hace nada sobre los 4 hotst). Dicho firewall sigue un conjunto de reglas que son explicadas en secciones posteriores del informe.

Por último, a modo de tener una simulación más personalizada, se dispuso de un archivo JSON con reglas y definiciones.

4. Implementación

Para poder implementar la simulación deseada se identificó que iban a ser necesario usar dos archivos, el primero es un pequeño script en Python que permite levantar la topología parametrizable, y el segundo, el programa Python que hace uso de **POX** para programar un controlador (se hace uso de la API).

4.1. Topología

Para la topología se creó el archivo **topology.py** en el cual definimos la estructura de la red simulada. Lo importante en este es que segun la cantidad de switches crea la cadena de los mismos para cumplir la consigna. Siempre en cada extremo hay dos hosts pero varia, según lo que se pida, cuantos enlaces son parte de la cadena entre extremo y extremo.

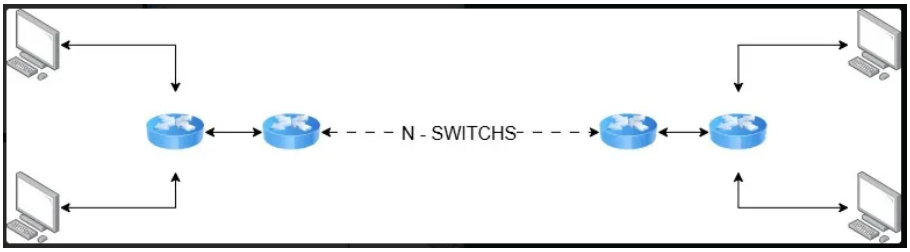

Figura 1: Representación de la Topología

Para ello iteramos sobre la cantidad de switches requerida, montando en los extremos los hosts y en el medio una unión de los switches.

4.2. Firewall

En esta sección, utilizamos las clases `Event` y la creación de un mensaje `ofp_flow_mod`. Este mensaje se usa para definir los atributos clave que deben considerarse al aplicar las reglas del firewall, como direcciones IP, protocolos o puertos. Por otro lado, la clase `Event` que se utiliza para gestionar los eventos que actúan como disparadores (*triggers*), permitiendo invocar las funciones correspondientes de forma dinámica cuando ocurre un evento relevante.

La implementación del firewall se basa en un modelo reactivo, donde se monitoriza el evento `ConnectionUp`. Este evento se genera cada vez que un nuevo elemento de la red, como un switch, establece conexión con el controlador. Cuando se detecta este evento, el controlador evalúa si el switch conectado tiene el identificador (`ID`) especificado en el archivo de configuración ya que las reglas se aplican a un único switch. Si coincide, se definen las reglas del firewell en dicho switch.

Este enfoque es eficiente porque evita configurar reglas en todos los switches de la red, lo que sería innecesario y podría consumir recursos de forma indebida. Al limitar la aplicación de reglas únicamente a los switches relevantes, se optimiza el rendimiento y se minimiza la sobrecarga en el controlador y los switches. Además, este método permite una fácil escalabilidad, ya que las reglas pueden definirse de manera selectiva en nuevos switches simplemente ajustando el archivo de configuración, sin necesidad de modificar el código del controlador.

Es importante destacar que este mecanismo garantiza que las reglas se apliquen de forma específica y controlada, manteniendo la flexibilidad necesaria para gestionar una red dinámica y en constante cambio. De esta forma, el firewall cumple su propósito sin afectar el tráfico o la funcionalidad en switches donde no es necesario.

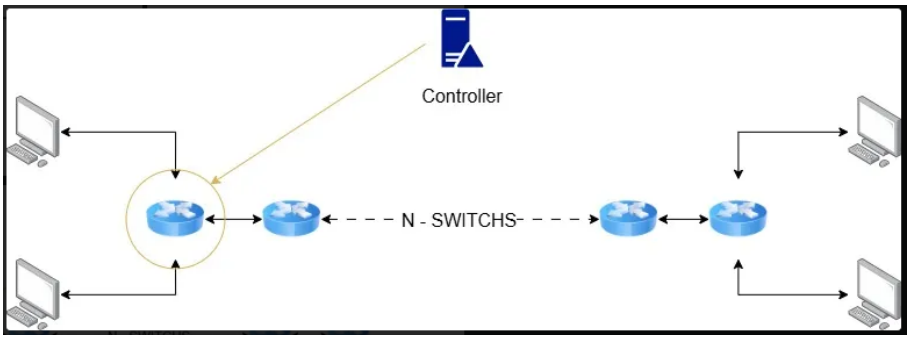

Figura 2: Representación del Firewall instalado en Switches

A partir de esto, se definen las tres reglas y el firewall queda configurado en el switch. Estas reglas se implementan utilizando la clase `ofp_flow_mod()` mencionada anteriormente, que permite crear mensajes personalizados para establecer reglas en los switches. Este método es clave, ya que permite configurar los atributos específicos de cada regla de manera precisa.

Es importante destacar que este proceso se realiza de manera independiente para cada regla, creando un nuevo mensaje tres veces, en lugar de generar una única regla que englobe todas las condiciones. Esto asegura que cada regla sea específica y modular, facilitando su mantenimiento y mejorando la claridad en la implementación del cortafuegos.

5. Ejecución

¡Es importante respetar el orden en que se levantan las distintas herramientas para evitar posibles problemas!

1. Levantar el firewall

En una terminal (`bash`), levantar el firewall con el siguiente comando:

```
python3 pox.py log.level --DEBUG openflow.of_01 forwarding.l2_learning firewall
```

2. Levantar la topología de red

En otra terminal (`bash`), levantar la topología de red deseada con el siguiente comando:

```
sudo mn --custom topology.py --topo tp2,ns=n --arp --switch ovsk --controller remot
```

ns=n indica la cantidad de switches usados para la red.

3. Levantar clientes y servidores

Se pueden levantar los clientes y servidores de la simulación usando la herramienta `Xterm` integrada en `Mininet` (levantarlos dentro de la terminal de mininet, es decir, en la shell generada por mininet):

```
xterm h1 h2 h3 h4
```

Se pueden levantar hasta 4 hosts (siendo uno el mínimo y cuatro el máximo). Es importante saber que lo que se hizo fue levantar simples terminales que representan a un host de Mininet. Para levantar los clientes y servidores va a ser necesrio usar Xterm.

Host	IP	Puerto
H1	10.0.0.1	Arbitrario a fines de la simulación
H2	10.0.0.2	Arbitrario a fines de la simulación
H3	10.0.0.3	Arbitrario a fines de la simulación
H4	10.0.0.4	Arbitrario a fines de la simulación

Cuadro 1: Configuración de hosts

Los hosts se levantan por defecto con las IPs especificadas en la tabla anterior. Respecto a los puertos, estos se especifican al momento de levantar algún cliente o servidor sobre un host.

4. Testear la configuración

Se recomienda testear la configuración seleccionada junto con el res de herramientas en ejecución (dentro de Mininet):

```
pingall
```

5. Levantar clientes y servidores

Servidor UDP Para levantar un servidor UDP en un host (una terminal de Xterm):

```
iperf -u -s -p <mi puerto>
```

- `-u`: UDP como protocolo de la capa de transporte.
- `-s`: El host actuará como servidor.
- `-p`: Puerto en el que se levantará el servidor.

Cliente UDP Para levantar un cliente UDP en un host (otra terminal de Xterm):

```
iperf -u -c <IP del servidor> -p <puerto del servidor>
```

- `-u`: UDP como protocolo de la capa de transporte.
- `-c`: El host actuará como cliente.
- `-p`: Puerto de destino al que se enviarán los mensajes.

Cliente TCP Para levantar un cliente TCP en un host (una terminal de Xterm):

```
iperf -c <IP del servidor> -p <puerto del servidor> -b <x> -n <x> -l <x> -t <x>
```

- `-c`: El host actuará como cliente.
- `-p`: Puerto de destino al que se enviarán los mensajes.
- `-b`: Limitación del ancho de banda (bits/s).
- `-n`: Limitación de la cantidad de paquetes.
- `-l`: Limitación del tamaño de los paquetes.
- `-t`: Limitación del tiempo de ejecución (tiempo de conexión).

Servidor TCP Para levantar un servidor TCP en un host (otra terminal de Xterm):

```
iperf -s -p <puerto> -b <x> -n <x> -l <x> -t <x>
```

- `-s`: El host actuará como servidor.
- `-p`: Puerto en el que se levantará el servidor.
- `-b`: Limitación del ancho de banda (bits/s).
- `-n`: Limitación de la cantidad de paquetes.
- `-l`: Limitación del tamaño de los paquetes.
- `-t`: Limitación del tiempo de ejecución (tiempo de conexión).

6. Pruebas

Para probar la simulación no se hizo más que poner a prueba las distintas reglas utilizadas para definir al firewall.

6.1. Regla 1

La regla 1 consiste en que se deben descartar todos los mensajes cuyo puerto destino sea 80. En las siguientes imágenes se puede observar la ejecución de un cliente y un servidor en el puerto 80 y como el primero a pesar de enviar varios paquetes nunca recibe un ACK ya que el servidor no registra la conexión

Figura 3: Screenshot de una terminal Iperf levantando un servidor en el puerto 80

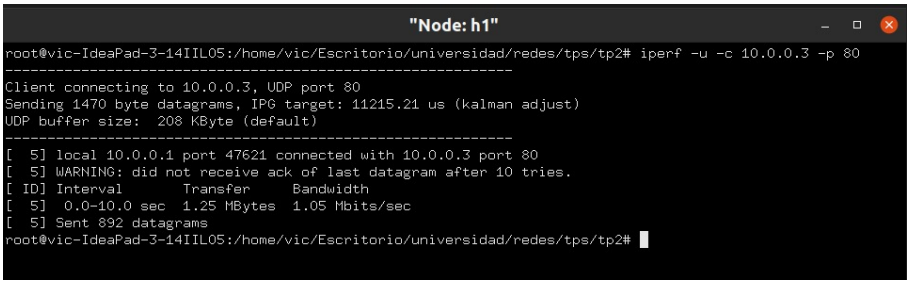

Figura 4: Screenshot de una terminal Iperf levantando un cliente

En la captura de Wireshark se puede observar como no se recibe ningún ACK luego de varios paquetes enviados

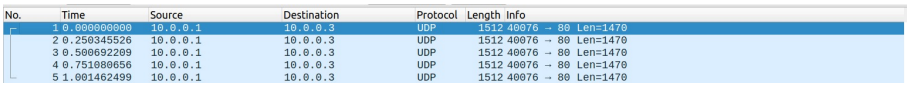

Figura 5: Screenshot de Wireshark

6.2. Regla 2

La regla 2 consiste en que se deben descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y estén utilizando el protocolo UDP. Siguiendo la misma lógica mencionada anteriormente, se puede observar en las imagenes el correcto funcionamiento

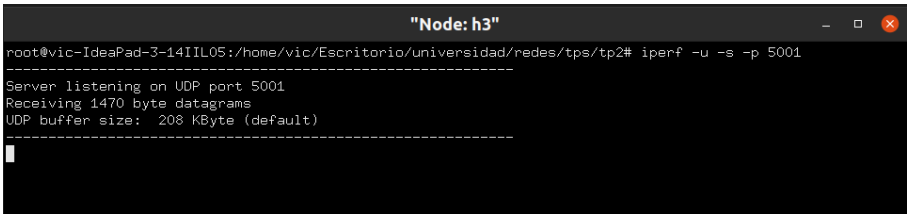

Figura 6: Screenshot de una terminal Iperf levantando un servidor en el puerto 5001

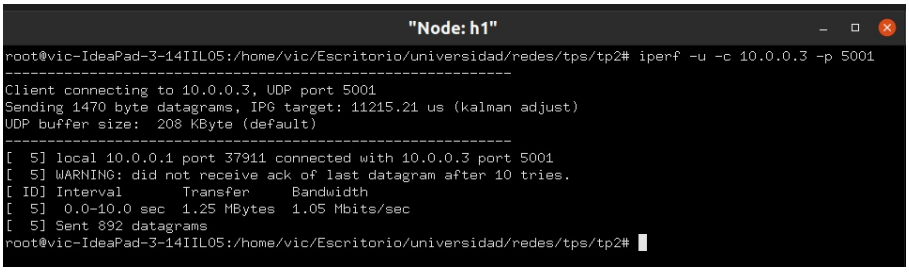

Figura 7: Screenshot de una terminal Iperf levantando un cliente UDP en el host 1

6.3. Regla 3

La regla 3 dice que, bajo dos hosts seleccionados, los mismos no deben poder comunicarse de ninguna forma. Para testear esta regla se hizo uso de la herramienta pingall donde se puede observar que, seleccionados los host 1 y 4, se detecta una conexión fallida tanto de h1 a h4 como de h4 a h1

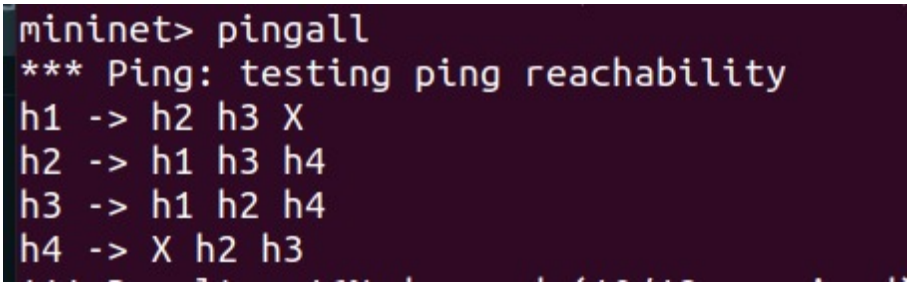

Figura 8: Screenshot de una terminal ejecutando pignall

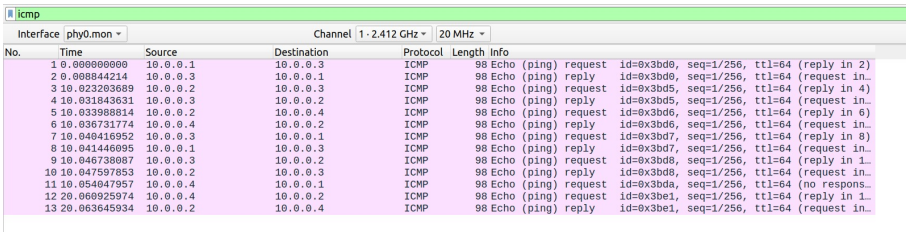

Figura 9: Screenshot de Wireshark en medio de un pingall

7. Preguntas a responder

7.1. ¿Cual es la diferencia entre un switch y un router? ¿Que tienen en comun?

La principal diferencia entre un *switch* y un *router* radica en cómo operan dentro de la red. Un *switch* trabaja en la **capa de enlace de datos** del modelo OSI y se encarga de interconectar dispositivos dentro de una misma red local (*LAN*), encaminando tramas basándose en direcciones MAC. Por otro lado, un *router* opera en la **capa de red** y permite la comunicación entre diferentes redes, utilizando direcciones IP para enrutar los paquetes.

Ambos dispositivos son esenciales para el transporte de datos y contribuyen a la conectividad en una red. Un *switch* asegura la comunicación eficiente entre dispositivos dentro de una red local, mientras que un *router* habilita la conexión entre distintas redes, como una red local e Internet.

En redes modernas, tanto *switches* como *routers* pueden incluir funciones avanzadas, como el control del tráfico, priorización de paquetes (*QoS*) y mecanismos de seguridad. Además, en entornos definidos por software, como OpenFlow, ambos dispositivos pueden programarse para realizar tareas específicas, ajustándose a las necesidades de la red y mejorando su flexibilidad.

7.2. ¿Cual es la diferencia entre un Switch convencional y un Switch OpenFlow?

La diferencia principal entre un *switch* convencional y un *switch OpenFlow* radica en cómo manejan el tráfico de red.

En los **switches convencionales**, el plano de datos y el plano de control están integrados internamente. Esto significa que las decisiones de encaminamiento y reenvío de paquetes se realizan dentro del propio *switch*, ya sea mediante hardware o software local. Además, la configuración de estos *switches* es individual y debe realizarse de manera anticipada, lo que genera un sistema rígido y difícil de gestionar, ya que cada dispositivo debe modificarse manualmente en caso de cambios.

Por otro lado, los **switches OpenFlow** ofrecen una arquitectura abierta y centralizada. En este caso, el control no está integrado en el *switch*, sino que es gestionado por un controlador externo dentro de un entorno de redes definidas por software (*Software Defined Networking*, SDN). Esto proporciona una gran flexibilidad, ya que el controlador SDN puede modificar en tiempo real el comportamiento del *switch*, adaptándolo dinámicamente a las necesidades de la red. Además, al centralizar el control, se facilita la gestión y se permite que la red responda de forma ágil y coordinada a situaciones de alta exigencia.

Esta diferencia representa una clara ventaja de los *switches OpenFlow* frente a los convencionales, ya que ofrecen una rápida respuesta y permiten la automatización en la gestión de la red, reduciendo la complejidad operativa y mejorando la capacidad de adaptación a cambios o necesidades específicas.

7.3. ¿Se pueden reemplazar todos los routers de la Intenet por Switches OpenFlow?

No, no es posible reemplazar todos los routers de Internet por *switches OpenFlow* debido a las funciones críticas que cumplen los routers en el escenario *inter-AS* (entre sistemas autónomos). Los routers operan en la **capa de red** y manejan protocolos fundamentales como BGP (*Border Gateway Protocol*), que es esencial para intercambiar información de enrutamiento entre diferentes sistemas autónomos y garantizar la conectividad global.

Por otro lado, aunque los *switches OpenFlow* son programables y ofrecen gran flexibilidad, están diseñados principalmente para tareas en redes locales o controladas, como entornos *intra-AS*. No cuentan con soporte nativo para protocolos *inter-AS* como BGP. Además, la complejidad y escala de Internet requieren capacidades avanzadas de agregación de rutas, gestión de políticas y resiliencia que los *switches OpenFlow* no pueden replicar completamente.

Por lo tanto, los routers siguen siendo indispensables para el funcionamiento de Internet a nivel global, especialmente en el contexto *inter-AS*, donde se requiere robustez, interoperabilidad y escalabilidad que van más allá de las capacidades actuales de los *switches OpenFlow*.

8. Dificultades encontradas

Las mayores dificultades encontradas estuvieron relacionadas con el aprendizaje de las herramientas necesarias para la simulación, las cuales eran desconocidas para el equipo de trabajo. Inicialmente, se dimensionó el trabajo práctico como más complejo de lo que resultó ser en realidad, debido a la falta de familiaridad con estas herramientas y al tiempo requerido para comprender su funcionamiento.

En particular, la conexión de un script de Python a Mininet utilizando los *flags* necesarios para levantar un controlador remoto programado con POX fue una de las dificultades específicas. Sin embargo, una vez configurado correctamente y comprendido el uso de las herramientas a partir de su documentación, fue posible avanzar de manera eficiente con el trabajo.

En cuanto a la implementación del firewall y sus reglas, se encontró un pequeño inconveniente al utilizar POX, especialmente al levantar un controlador que gestionara la red simulada por Mininet. A pesar de estos retos iniciales, la integración entre ambas herramientas se logró con éxito, lo que permitió completar la implementación propuesta.

9. Conclusión

El trabajo realizado resultó ser muy interesante y permitió profundizar en el funcionamiento de un *firewall*, así como en su rol dentro de los elementos de la red, particularmente los *switches*, en el contexto de este trabajo práctico. Además, se logró demostrar la relevancia de protocolos como OpenFlow para la gestión eficiente de una subred y la flexibilidad que ofrecen para implementar soluciones personalizadas en entornos definidos por software.

Así mismo, el uso de simulaciones resultó ser un recurso clave al momento de programar un controlador remoto, permitiendo identificar y resolver posibles problemas antes de desplegar soluciones en entornos reales. Este enfoque no solo facilita el aprendizaje, sino que también asegura un diseño más robusto y confiable para futuras implementaciones.