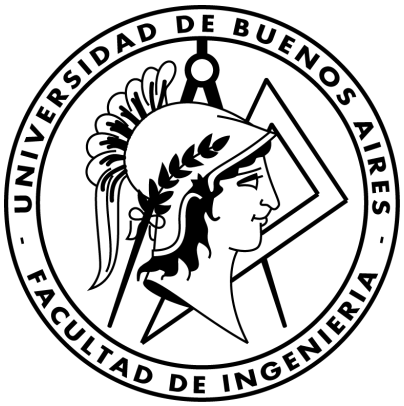
Trabajo Práctico 2: Software-Defined Networks

Facultad de Ingeniería de la Universidad de Buenos Aires

Redes

Cátedra Hamelin-Lopez Pecora



Demarchi, Ignacio
Padrón: 107835
email: idemarchi@fi.uba.ar

Lijs, Theo
Padrón: 109472
email: tlijs@fi.uba.ar

Schneider, Valentin
Padrón: 107964
email: vschneider@fi.uba.ar

Orsi, Tomas Fabrizio
Padrón: 109735
email: torsi@fi.uba.ar

Contents

1 Introducción

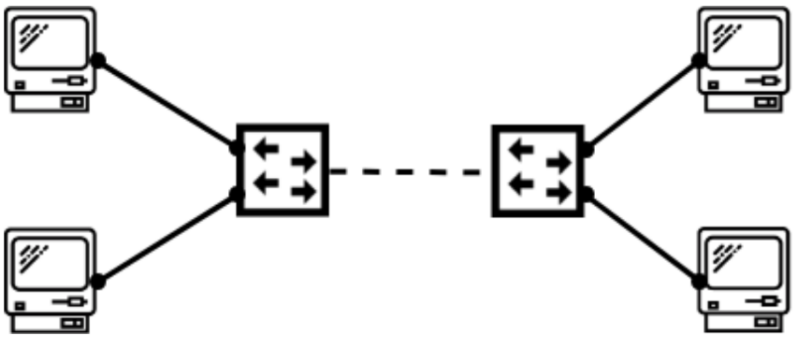
En este trabajo práctico se implementó un SDN que, mediante OpenFlow (utilizando POX), implementa un Firewall sobre una red creada en Mininet. Para ver el programa en acción y acercar la simulación a un caso de uso real, dentro de los hosts de la red de Mininet se utiliza iperf para establecer fácilmente una conexión entre clientes y servidores y observar el funcionamiento del Firewall en acción. Para comprobar esto, se utiliza Wireshark, donde se observan los paquetes siendo enviados.

2 Herramientas utilizadas

A continuación se detalla el uso de cada herramienta mencionada para elaborar el trabajo práctico.

2.1 Mininet

Para utilizar Mininet, la topología se define en `mytopo.py`. La misma recibe como parámetro la cantidad de switches a utilizar.



Al correr el comando para levantar Mininet, se establece la IP del controlador que se va a utilizar. Esto es para que, luego, cuando corramos el controlador, el mismo pueda modificar los switches de la topología y maneje el control plane de la red de Mininet.

2.2 POX - WIP

Para implementar el controlador con OpenFlow, se utilizó la biblioteca POX. El controlador utiliza L2 learning para que los switches aprendan automáticamente a reenviar paquetes.

Para el firewall implementamos el metodo `_handle_ConnectionUp` que se encarga de instalar las reglas en el switch designado como firewall. Para esto, se lee el archivo `policies.json` que contiene las reglas a aplicar y se las traduce

a objetos del tipo `ofp_match`. Este objeto es un conjunto de criterios que se utilizan para identificar flujos de red en el contexto de utilizacion de OpenFlow.

Luego entonces cuando un paquete llega al switch, se verifica si cumple con alguna de las reglas del firewall. En caso de cumplir con alguna regla, se descarta el paquete. En caso contrario, se lo deja pasar.

2.2.1 POX en ejecucion

A continuación se observa lo que el controlador registra al iniciarse:

```
valen1611@ubuntu1611:~/code/redes$ make run
sed -i 's/IPDELSWITCHCONELFIREWALL = [0-9]/IPDELSWITCHCONELFIREWALL = 1/' src/firewall.py
pox/pox.py firewall forwarding.l2_learning log.level --DEBUG samples.pretty_log
POX 0.6.0 (fangtooth) / Copyright 2011-2018 James McCauley, et al.
[core                  ] POX 0.6.0 (fangtooth) going up...
[core                  ] Running on CPython (2.7.18/Oct 15 2023 16:43:11)
[core                  ] Platform is Linux-6.5.0-1024-oem-x86_64-with-Ubuntu-22.04-jammy
[core                  ] POX 0.6.0 (fangtooth) is up.
[openflow.of_01        ] Listening on 0.0.0.0:6633
[openflow.of_01        ] [00-00-00-00-00-07 1] connected
[forwarding.l2_learning ] Connection [00-00-00-00-00-07 1]
[openflow.of_01        ] [00-00-00-00-00-01 2] connected
[firewall              ] Firewall rules installed on 00-00-00-00-00-01
[firewall              ] Installing policy: ofp_match
  dl_type: 0x800
  nw_src: 10.0.0.1
  nw_dst: 10.0.0.3
[firewall              ] Installing policy: ofp_match
  dl_type: 0x800
  nw_src: 10.0.0.3
  nw_dst: 10.0.0.1
[firewall              ] Installing policy: ofp_match
  dl_type: 0x800
  nw_proto: 17
  nw_src: 10.0.0.1
  tp_dst: 5001
[firewall              ] Installing policy: ofp_match
  dl_type: 0x800
  nw_proto: 6
  tp_dst: 80
[firewall              ] Installing policy: ofp_match
  dl_type: 0x800
  nw_proto: 17
  tp_dst: 80
[forwarding.l2_learning ] Connection [00-00-00-00-00-01 2]
[openflow.of_01        ] [00-00-00-00-00-05 7] connected
[forwarding.l2_learning ] Connection [00-00-00-00-00-05 7]
[openflow.of_01        ] [00-00-00-00-00-06 4] connected
[forwarding.l2_learning ] Connection [00-00-00-00-00-06 4]
[openflow.of_01        ] [00-00-00-00-00-02 6] connected
[forwarding.l2_learning ] Connection [00-00-00-00-00-02 6]
[openflow.of_01        ] [00-00-00-00-00-04 3] connected
[forwarding.l2_learning ] Connection [00-00-00-00-00-04 3]
[openflow.of_01        ] [00-00-00-00-00-03 5] connected
[forwarding.l2_learning ] Connection [00-00-00-00-00-03 5]
```

Figure 1: Pox mostrando las reglas que instala

2.3 Wireshark & iperf

Para comprobar el correcto funcionamiento de la red y del Firewall, se utiliza iperf para simular clientes y servidores sin tener que configurarlos manualmente en los hosts de Mininet.

Por otro lado utilizamos Wireshark para escuchar los paquetes que se envían en la red y verificar que el Firewall está funcionando correctamente.

A continuación se muestra el correcto funcionamiento del controlador, usando `pingall` dentro de Mininet y escuchando con Wireshark.

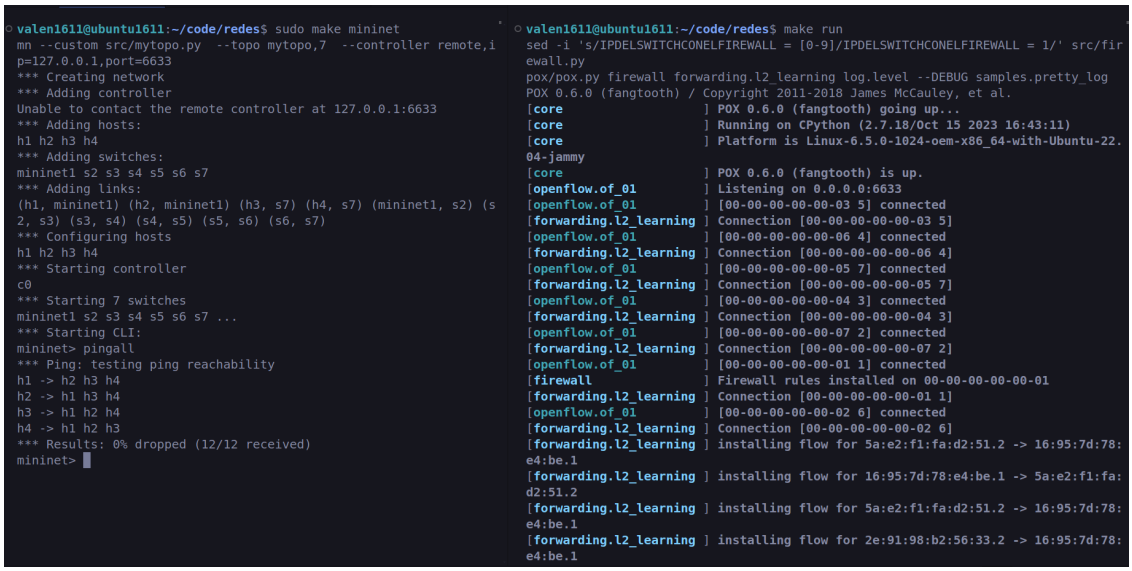


IMAGEN TRAZA WIRESHARK CON EL PINGALL DE MININET

Para comprobar que no solo funciona con ICMP, utilizamos iperf. Con iperf simulamos clientes y servidores. En el ejemplo a continuación, el host h1 actúa como cliente y el host h3 actúa como servidor, comunicándose por TCP.

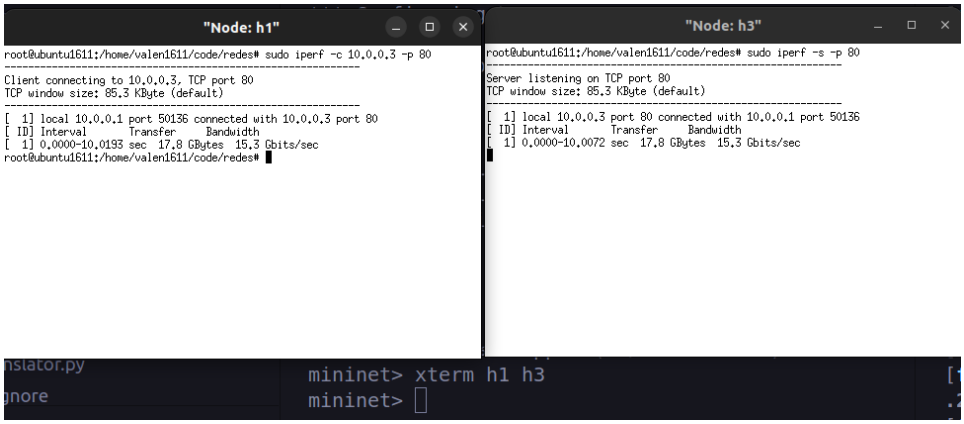


IMAGEN TRAZA WIRESHARK CON EL IPERF BASICO

3 Resultados de simulaciones - WIP IMAGENES

3.1 Puerto Destino 80

Simulación para descartar todos los mensajes cuyo puerto destino sea 80.

3.1.1 Reglas

```
{
    "policies":[
        {
            "dst_port": "80"
        }
    ]
}
```

3.1.2 Wireshark

3.1.3 Logs del controlador

3.2 Host 1, Puerto 5001 y UDP

Simulación para descartar todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y utilicen el protocolo UDP.

3.2.1 Reglas

```
{
    "policies":[
        {
            "src_ip": "10.0.0.1",
            "dst_port": 5001,
            "protocol": "UDP"
        }
    ]
}
```

3.2.2 Wireshark

3.2.3 Logs del controlador

3.3 Dos hosts no se comunican entre sí

Simulación donde se eligen dos hosts cualquiera, y los mismos no pueden comunicarse de ninguna forma.

3.3.1 Reglas

```
{
    "policies":[
        {
            "banned_tuples": ["10.0.0.1", "10.0.0.3"]
        }
    ]
}
```

3.3.2 Wireshark

3.3.3 Logs del controlador

4 Preguntas a responder

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

La principal diferencia es que un switch opera en la capa 2 (enlace) y un router en la capa 3 (red). Los switches redireccionan utilizando la dirección MAC de los dispositivos, mientras que los routers utilizan la IP.

Lo que tienen en común es que ambos funcionan para redireccionar paquetes y permitir que hosts en distintas partes del mundo puedan comunicarse entre sí.

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

La diferencia más importante entre un Switch convencional y uno OpenFlow es que el OpenFlow puede ser gestionado mediante software con un controlador centralizado, lo que permite automatizar y agilizar el proceso. Los switches convencionales no tienen el plano de control y el de datos desacoplados, por lo que configurarlos requiere más trabajo.

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

En principio, se podría, más allá de incompatibilidades que podrían llegar a ser parcheadas. Pero no es algo factible realmente. Surgirían muchísimos problemas de seguridad, rendimiento, interoperabilidad, etc. En el contexto de interASes, si bien al manejar todo desde una misma tecnología podría facilitar, por ejemplo, la implementación de políticas de tráfico para mejorar la flexibilidad de la gestión del mismo, se presentaría un single point of failure ya que todo dependería de OpenFlow. Una vulnerabilidad que se descubra sobre el protocolo en sí dejaría expuesto a todo el Internet global. Y dentro de una red privada, si no se tienen routers no podríamos utilizar NATs de la forma actual, habría que implementarla con los Switches.

5 Dificultades encontradas - WIP

6 Conclusión - WIP