

INTRODUCCIÓN A LOS SISTEMAS DISTRIBUIDOS (75.43)

Trabajo Práctico N°2: SOFTWARE-DEFINED NETWORKS

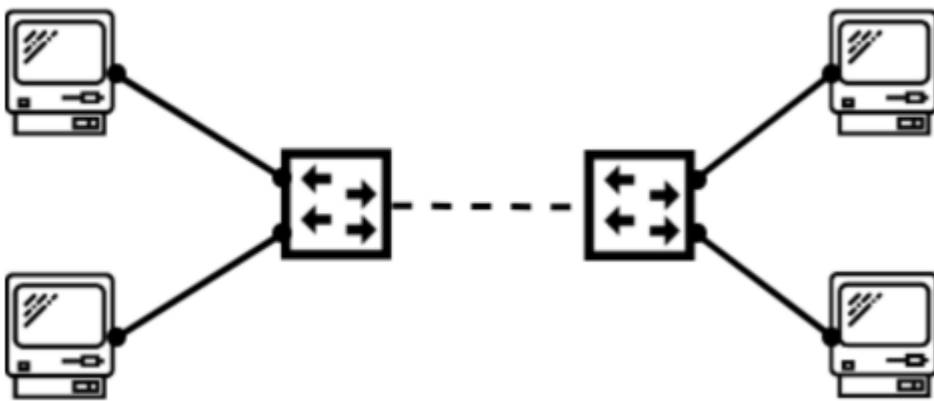
Alumno	Padrón
Camila Fernández Marchitelli	102515
Lucas Aldazabal	107705
Bautista Boselli	107490
Alejo Fábregas	106160
Camilo Fábregas	103740

INTRODUCCIÓN

El objetivo del trabajo práctico es la construcción de una topología dinámica y parametrizable, utilizando el protocolo 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.

IMPLEMENTACIÓN

Se elaboró un código capaz de generar la **topología** indicada, cumpliendo con la parametrización de ciertas variables como los switches. Se recibe por parámetro la cantidad de switches, formando una cadena en cuyos extremos se tienen dos hosts. La topología implementada es similar a la de la imagen, donde en el centro se representa el número variable de switches, además de los 4 hosts.



Para el **firewall** utilizamos la biblioteca POX, y se implementaron las reglas solicitadas en el enunciado:

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

Las reglas fueron definidas en un archivo JSON que es parseado desde pox.py al inicializar el controlador.

Configuración del firewall

El formato elegido es el siguiente:

```
{
  "block_communications": [ ["10.10.10.2", "10.10.11.1"] ],
  "block_messages": [
    {
      "dest_port": 80,
      "protocol": "TCP"
    },
    {
      "dest_port": 80,
      "protocol": "UDP"
    },
    {
      "src_ip": "10.10.10.1",
      "dest_port": 5001,
      "protocol": "UDP"
    }
  ]
}
```

Donde cada campo representa:

- block_communications: contiene una lista de tuplas con las dos IP de los dos hosts para bloquear la comunicación entre ellos.
- block_messages: lista de reglas del firewall. Las primeras dos corresponden a la regla 1 (bloquear puerto 80 TCP/UDP), y la tercera entrada corresponde a la regla 2.

Las reglas de **block_messages** pueden tener los siguientes datos:

Nombre	Descripción	Requerido
protocol	protocolo del mensaje (UDP/TCP)	Si
src_ip	IP de origen	No
dest_ip	IP de destino	No
src_port	Puerto de origen	No
dest_port	Puerto de destino	No

Ejemplo de reglas.json:

```
{
  "block_messages": [
    {
      "protocol": "UDP", // requerido
      "src_ip": "10.10.10.1",
      "dest_ip": "10.10.11.2",
      "src_port": 4000,
      "dest_port": 80
    }
  ]
}
```

Configuración de la topología

Existen dos configuraciones a nivel de la topología de la red:

- `cant_switches`: cantidad de switches en la hilera
- `switch_firewall`: el número del switch que tendrá las reglas de firewall

Ejemplo de config.json

```
{
  "cant_switches": 10,
  "switch_firewall": 3
}
```

Iniciar POX

Ejecutar los siguientes comandos antes de iniciar mininet:

```
> cp firewall.py [PATH_POX]/pox/misc/firewall.py
```

En una terminal en el directorio del proyecto

```
> ./pox/pox.py log.level forwarding.l2_learning misc.firewall
```

En una terminal en el directorio del proyecto

Iniciar Mininet

```
> sudo python3 scripts.py [cli|test1|test2|test3|test4]
```

En una terminal en el directorio del proyecto

PREGUNTAS A RESPONDER

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

Ambos dispositivos tienen en común que pueden conectar dispositivos entre sí, además de poder decidir la salida de un paquete en función de sus direcciones. Ambos poseen un plano de control y un plano de datos.

El switch solo puede conectar dispositivos dentro de una misma red, determinando un puerto de salida a partir de la dirección MAC destino. El router puede conectar dispositivos pertenecientes a distintas redes, ya que utiliza la dirección IP para definir el puerto de salida. Por eso, los routers permiten no solo conectar switches entre sí, sino también conectar a un dispositivo con el resto de Internet.

Los routers pueden funcionar en redes con ciclos, ya que utilizan la estructura jerárquica de la dirección IP (además del header). En cambio, es requisito que no haya ciclos en una red de switches (ya que no hay jerarquía en las direcciones MAC), de lo contrario habrá una retransmisión infinita de paquetes.

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

La principal diferencia entre un switch convencional y un switch OpenFlow es que en un switch convencional los planos de datos y de control se ubican en el mismo dispositivo.

En un switch OpenFlow lo que ocurre es que el plano de control está ubicado en un controlador externo, y por lo tanto su funcionalidad es ajena al switch. Únicamente el plano de datos está ubicado en el switch OpenFlow.

OpenFlow, al ser un protocolo, es lo que permite la comunicación entre ambos planos. Lo que se logra con esto es una retransmisión basada en flujos, utilizando información de los headers de capa superior, capa de red y capa de transporte.

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

Considerando el escenario interASes, eso implica que los routers tengan que implementar el protocolo BGP: y esto significa que cada dispositivo debería tener almacenada una tabla con muchísimas entradas.

Además, en el caso de que toda la Internet utilizara switches OpenFlow, la idea de tener un plano de control externo consultado por todos estos dispositivos traería distintas complicaciones dado a la masividad de la red.

Por estos motivos es que se dice que las Software-Defined Networks no escalan bien para lo que es Internet, sino que se recomiendan para redes más pequeñas (como lo pueden ser los sistemas autónomos).

HIPÓTESIS Y SUPUESTOS REALIZADOS

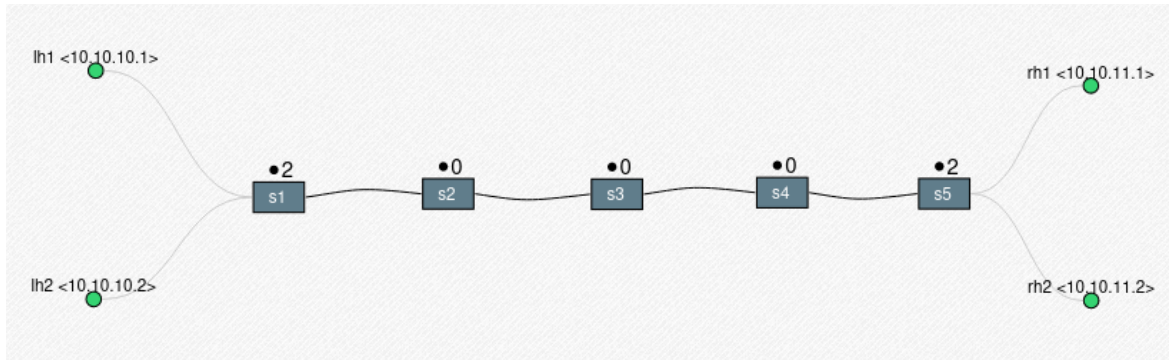
- Para cada regla que corresponda con bloquear un determinado puerto, es obligatorio aclarar el protocolo de transporte (ya sea TCP como UDP). Si se quisiese bloquear todas las comunicaciones independientemente del protocolo, se puede crear 2 reglas bloqueando el puerto una con cada protocolo.
- Cuando se elige un switch que no se encuentra en uno de los extremos, los dos hosts que se eligen para bloquear la comunicación deben estar en lados opuestos de la cadena de switches ya que sino no atravesarían el switch con el firewall y si podrían comunicarse.

PRUEBAS

Visualizador de topologías

Podemos visualizar nuestra topología custom tipo cadena con el visualizador de topologías de Mininet:

<http://demo.spear.narmox.com/app/?apiurl=demo#!/mininet>



Para obtener esta visualización, en Mininet obtenemos la siguiente información:

```
mininet> dump
<Host lh1: lh1-eth0:10.10.10.1 pid=20775>
<Host lh2: lh2-eth0:10.10.10.2 pid=20777>
<Host rh1: rh1-eth0:10.10.11.1 pid=20779>
<Host rh2: rh2-eth0:10.10.11.2 pid=20781>
<OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None,s1-eth3:None pid=20786>
<OVSSwitch s2: lo:127.0.0.1,s2-eth1:None,s2-eth2:None pid=20789>
<OVSSwitch s3: lo:127.0.0.1,s3-eth1:None,s3-eth2:None pid=20792>
<OVSSwitch s4: lo:127.0.0.1,s4-eth1:None,s4-eth2:None pid=20795>
<OVSSwitch s5: lo:127.0.0.1,s5-eth1:None,s5-eth2:None,s5-eth3:None pid=20798>
<RemoteController c0: 127.0.0.1:6633 pid=20767>
```

```
mininet> links
lh1-eth0<->s1-eth2 (OK OK)
lh2-eth0<->s1-eth3 (OK OK)
s1-eth1<->s2-eth1 (OK OK)
s2-eth2<->s3-eth1 (OK OK)
s3-eth2<->s4-eth1 (OK OK)
s4-eth2<->s5-eth1 (OK OK)
s5-eth2<->rh1-eth0 (OK OK)
s5-eth3<->rh2-eth0 (OK OK)
```

Wireshark:

En un pingall, dos hosts particulares no pueden comunicarse y el resto si

No.	Time	Source	Destination	Protocol	Length	Info
4580	27.228265173	10.10.11.2	10.10.10.2	ICMP	100	Echo (ping) reply id=0x8c3d, seq=1/256, ttl=64
4581	27.2303152099	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4586	27.233317068	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4587	27.233318089	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4592	27.236120413	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4593	27.236122273	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4598	27.239889578	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4599	27.239892575	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4604	27.245590702	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4605	27.245593969	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4611	27.249068697	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4612	27.249071147	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4617	27.252491365	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4618	27.252493452	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4623	27.256213626	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4624	27.256219475	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4629	27.261607881	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4630	27.261609614	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4635	27.264560573	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4636	27.264562082	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (no response found!)
4641	27.267988164	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) request id=0x5113, seq=1/256, ttl=64 (reply in 4642)
4642	27.267973168	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64 (request in 4641)
4647	27.269868769	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4648	27.269871767	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4655	27.274498668	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4656	27.274501907	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4663	27.277608196	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4664	27.277681880	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4670	27.279685677	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4671	27.279687568	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4677	27.283989095	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4678	27.283994781	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4684	27.286826289	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4685	27.286828168	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4690	27.289948059	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4691	27.289951672	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4697	27.293853718	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4698	27.293855544	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4704	27.297734533	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4705	27.297736760	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4710	27.301591276	10.10.11.1	10.10.10.1	ICMP	100	Echo (ping) reply id=0x5113, seq=1/256, ttl=64
4712	27.304580039	10.10.11.1	10.10.10.2	ICMP	100	Echo (ping) request id=0x9428, seq=1/256, ttl=64 (no response found!)
4717	27.306585828	10.10.11.1	10.10.10.2	ICMP	100	Echo (ping) request id=0x9428, seq=1/256, ttl=64 (no response found!)
4718	27.306586233	10.10.11.1	10.10.10.2	ICMP	100	Echo (ping) request id=0x9428, seq=1/256, ttl=64 (no response found!)
4723	27.309450126	10.10.11.1	10.10.10.2	ICMP	100	Echo (ping) request id=0x9428, seq=1/256, ttl=64 (no response found!)
4724	27.309451606	10.10.11.1	10.10.10.2	ICMP	100	Echo (ping) request id=0x9428, seq=1/256, ttl=64 (no response found!)
4730	27.312915897	10.10.11.1	10.10.10.2	ICMP	100	Echo (ping) request id=0x9428, seq=1/256, ttl=64 (no response found!)

```
camila@ubuntu:~/Documents/distribuidos/tp_sdn_distribuidos$ sudo python3 topologia.py
Creando Topologia
*** 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:
lh1 lh2 rh1 rh2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(lh1, s1) (lh2, s1) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9) (s9, s10) (s10, rh1) (s10, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2
Enter para Iniciar Mininet
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
lh1 lh1-eth0:s1-eth2
lh2 lh2-eth0:s1-eth3
rh1 rh1-eth0:s10-eth2
rh2 rh2-eth0:s10-eth3
Enter para Ejecutar pingall
*** Ping: testing ping reachability
lh1 -> lh2 rh1 rh2
lh2 -> lh1 X rh2
rh1 -> lh1 X rh2
rh2 -> lh1 lh2 rh1
*** Results: 16% dropped (10/12 received)
*** Starting CLI:
mininet>
```

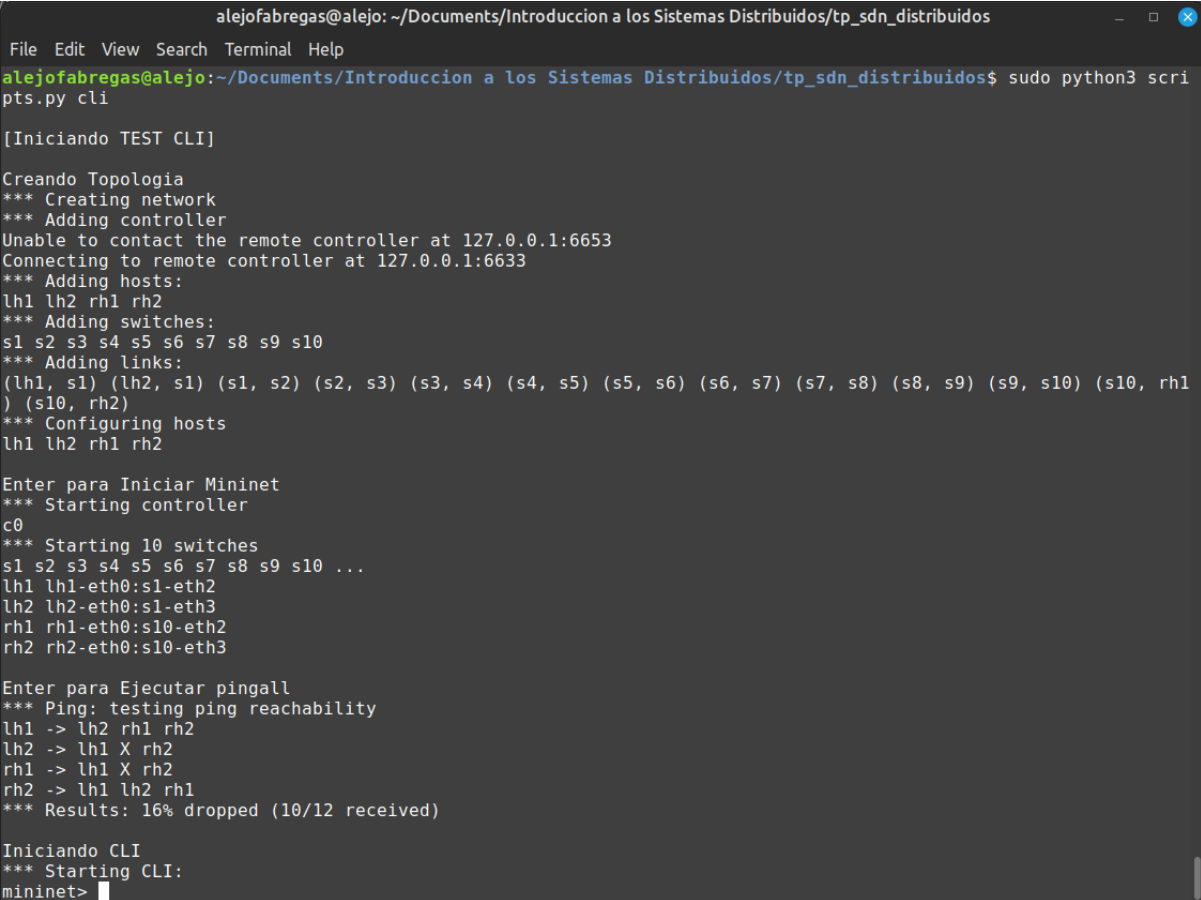

Scripts

Tenemos configurados 5 tipos de scripts para hacer tests semi automatizados de la topología y el firewall.

Acceder a CLI (Default)

Hace un pingall y luego se accede a la CLI de mininet.

```
> sudo python3 scripts.py cli
```



```
alejofabregas@alejo: ~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos
File Edit View Search Terminal Help
alejofabregas@alejo:~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos$ sudo python3 scri
pts.py cli

[Iniciando TEST CLI]

Creando Topologia
*** 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:
lh1 lh2 rh1 rh2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(lh1, s1) (lh2, s1) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9) (s9, s10) (s10, rh1
) (s10, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2

Enter para Iniciar Mininet
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
lh1 lh1-eth0:s1-eth2
lh2 lh2-eth0:s1-eth3
rh1 rh1-eth0:s10-eth2
rh2 rh2-eth0:s10-eth3

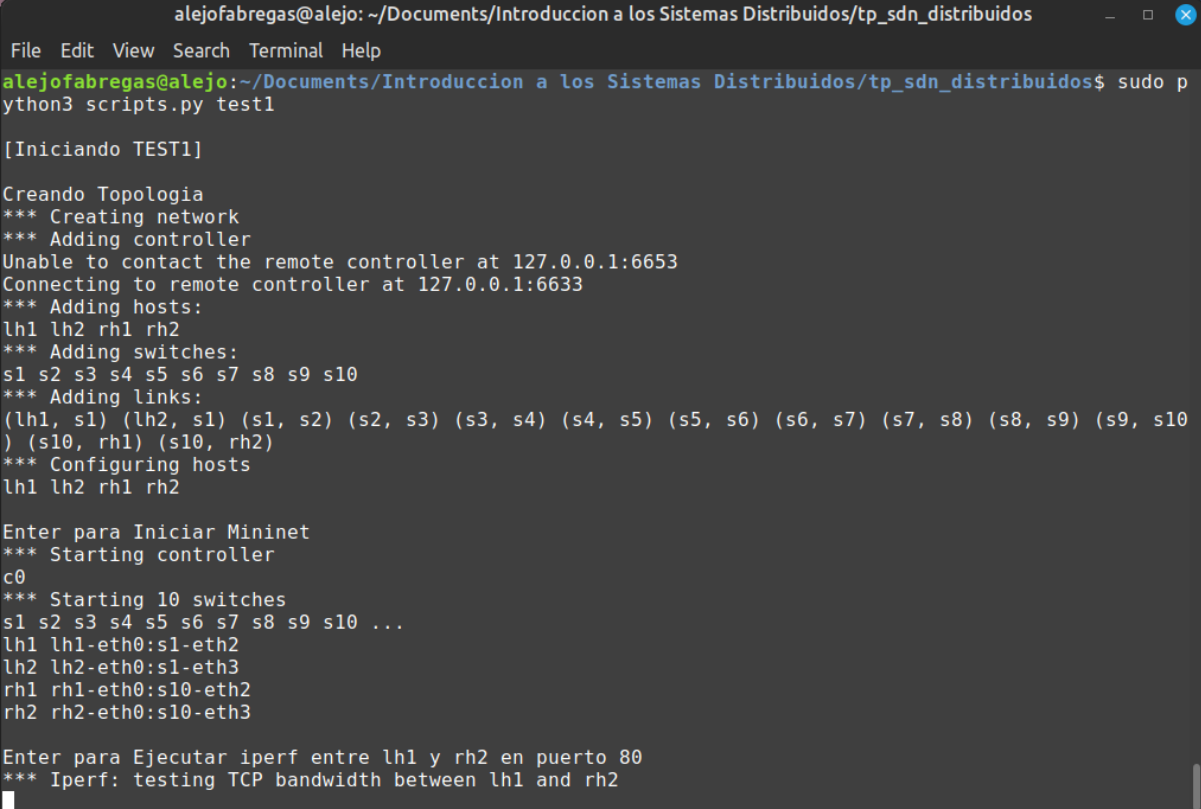
Enter para Ejecutar pingall
*** Ping: testing ping reachability
lh1 -> lh2 rh1 rh2
lh2 -> lh1 X rh2
rh1 -> lh1 X rh2
rh2 -> lh1 lh2 rh1
*** Results: 16% dropped (10/12 received)

Iniciando CLI
*** Starting CLI:
mininet> 
```

Test regla 1

Ejecutar un iperf entre lh1 y rh2 en el puerto 80 para validar que se descartan todos los mensajes con puerto destino 80.

```
> sudo python3 scripts.py test1
```



```
alejofabregas@alejo: ~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos
File Edit View Search Terminal Help
alejofabregas@alejo:~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos$ sudo python3 scripts.py test1

[Iniciando TEST1]

Creando Topologia
*** 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:
lh1 lh2 rh1 rh2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(lh1, s1)(lh2, s1)(s1, s2)(s2, s3)(s3, s4)(s4, s5)(s5, s6)(s6, s7)(s7, s8)(s8, s9)(s9, s10)
(s10, rh1)(s10, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2

Enter para Iniciar Mininet
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
lh1 lh1-eth0:s1-eth2
lh2 lh2-eth0:s1-eth3
rh1 rh1-eth0:s10-eth2
rh2 rh2-eth0:s10-eth3

Enter para Ejecutar iperf entre lh1 y rh2 en puerto 80
*** Iperf: testing TCP bandwidth between lh1 and rh2
```

Test regla 2

Ejecutar un iperf desde lh1 en puerto 5001 y UDP para validar que se descartan todos los mensajes que provengan del host 1, tengan como puerto destino el 5001, y estén utilizando el protocolo UDP.

```
> sudo python3 scripts.py test2
```

```
alejofabregas@alejo: ~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos
File Edit View Search Terminal Help
alejofabregas@alejo:~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos$ sudo python3 scri
pts.py test2

[Iniciando TEST2]

Creando Topologia
*** 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:
lh1 lh2 rh1 rh2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(lh1, s1) (lh2, s1) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9) (s9, s10) (s10, rh1
) (s10, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2

Enter para Iniciar Mininet
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
lh1 lh1-eth0:s1-eth2
lh2 lh2-eth0:s1-eth3
rh1 rh1-eth0:s10-eth2
rh2 rh2-eth0:s10-eth3

Enter para Ejecutar iperf desde el lh1 en puerto 5001 y UDP
*** Iperf: testing UDP bandwidth between lh1 and rh1
could not parse iperf output: -----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----
*** Results: ['10M', '', '10.5 Mbits/sec']

Iniciando CLI
*** Starting CLI:
mininet> 
```

Test regla 3

Ejecutar iperf entre lh2 y rh1 para validar que dados dos hosts cualquiera (configurables vía reglas) no se puedan comunicar de ninguna forma.

```
> sudo python3 scripts.py test3
```

```
alejofabregas@alejo: ~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos
File Edit View Search Terminal Help
alejofabregas@alejo:~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos$ sudo python3 scri
pts.py test3

[Iniciando TEST3]

Creando Topologia
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
lh1 lh2 rh1 rh2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(lh1, s1) (lh2, s1) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9) (s9, s10) (s10, rh1
) (s10, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2

Enter para Iniciar Mininet
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
lh1 lh1-eth0:s1-eth2
lh2 lh2-eth0:s1-eth3
rh1 rh1-eth0:s10-eth2
rh2 rh2-eth0:s10-eth3

Enter para Ejecutar PingAll
*** Ping: testing ping reachability
lh1 -> lh2 rh1 rh2
lh2 -> lh1 X rh2
rh1 -> lh1 X rh2
rh2 -> lh1 lh2 rh1
*** Results: 16% dropped (10/12 received)

Enter para Ejecutar iperf entre lh2 y rh1
*** Iperf: testing TCP bandwidth between lh2 and rh1
```

Test hosts mismo lado

Ejecutar iperf entre lh1 y lh2 para ver que dos hosts del mismo lado pueden comunicarse entre sí por puerto 80 (prohibido por regla 1) dependiendo del `switch_firewall` del `config.json`:

- si el `switch_firewall` es el 1 (switch del extremo donde conectan estos hosts): La regla 1 prohíbe la comunicación
- si el `switch_firewall` es cualquier otro: Se pueden comunicar por no pasar por el firewall

```
> sudo python3 scripts.py test4
```

En este primer caso, corremos el `test4` con el `switch_firewall` en 1, por lo que el switch está conectado a ambos hosts, bloqueando la comunicación. Por eso se ve que el iperf no termina.

```
alejofabregas@alejo: ~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos
File Edit View Search Terminal Help
alejofabregas@alejo:~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos$ sudo python3 scri
pts.py test4

[Iniciando TEST1]

Creando Topologia
*** 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:
lh1 lh2 rh1 rh2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(lh1, s1) (lh2, s1) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9) (s9, s10) (s10, rh1)
(s10, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2

Enter para Iniciar Mininet
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
lh1 lh1-eth0:s1-eth2
lh2 lh2-eth0:s1-eth3
rh1 rh1-eth0:s10-eth2
rh2 rh2-eth0:s10-eth3

Enter para Ejecutar iperf entre lh1 y lh2 en puerto 80
*** Iperf: testing TCP bandwidth between lh1 and lh2
```

En este otro caso, corremos el test4 con el switch_firewall en 3, por lo que el switch no está conectado a ambos hosts, sino que está en el medio de la cadena. Por eso la comunicación entre estos hosts no se bloquea y el iperf termina, mostrando los resultados.

```
alejofabregas@alejo: ~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos
File Edit View Search Terminal Help
alejofabregas@alejo:~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos$ sudo python3 scri
pts.py test4

[Iniciando TEST1]

Creando Topologia
*** 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:
lh1 lh2 rh1 rh2
*** Adding switches:
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10
*** Adding links:
(lh1, s1) (lh2, s1) (s1, s2) (s2, s3) (s3, s4) (s4, s5) (s5, s6) (s6, s7) (s7, s8) (s8, s9) (s9, s10) (s10, rh1
) (s10, rh2)
*** Configuring hosts
lh1 lh2 rh1 rh2

Enter para Iniciar Mininet
*** Starting controller
c0
*** Starting 10 switches
s1 s2 s3 s4 s5 s6 s7 s8 s9 s10 ...
lh1 lh1-eth0:s1-eth2
lh2 lh2-eth0:s1-eth3
rh1 rh1-eth0:s10-eth2
rh2 rh2-eth0:s10-eth3

Enter para Ejecutar iperf entre lh1 y lh2 en puerto 80
*** Iperf: testing TCP bandwidth between lh1 and lh2
*** Results: ['14.0 Gbits/sec', '14.0 Gbits/sec']

Iniciando CLI
*** Starting CLI:
mininet>
```

Controlador POX

Al iniciar el controlador POX y luego Mininet, se puede ver cómo los switches se conectan al controlador, y éste define las reglas indicadas en el archivo reglas.json únicamente para el switch definido el archivo config.json.

```

alejofabregas@alejo: ~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distrib...
File Edit View Search Terminal Help

alejofabregas@alejo:~/Documents/Introduccion a los Sistemas Distribuidos/tp_sdn_distribuidos$ ./pox/pox.py log.level forwarding.l2_learning misc.firewall
POX 0.6.0 (fangtooth) / Copyright 2011-2018 James McCauley, et al.
INFO:misc.firewall:Iniciando controlador POX
INFO:core:POX 0.6.0 (fangtooth) is up.
INFO:openflow.of_01:[00-00-00-00-00-07 7] connected
INFO:misc.firewall:[SWITCH dpid: 7] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-04 9] connected
INFO:misc.firewall:[SWITCH dpid: 4] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-10 4] connected
INFO:misc.firewall:[SWITCH dpid: 16] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-06 1] connected
INFO:misc.firewall:[SWITCH dpid: 6] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
INFO:misc.firewall:[SWITCH dpid: 3] Conectado con exito al controlador
INFO:misc.firewall:[SWITCH dpid: 3] Configurando las reglas para este switch
INFO:openflow.of_01:[00-00-00-00-00-09 8] connected
INFO:misc.firewall:[SWITCH dpid: 9] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
INFO:misc.firewall:[SWITCH dpid: 1] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-05 10] connected
INFO:misc.firewall:[SWITCH dpid: 5] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-08 5] connected
INFO:misc.firewall:[SWITCH dpid: 8] Conectado con exito al controlador
INFO:openflow.of_01:[00-00-00-00-00-02 6] connected
INFO:misc.firewall:[SWITCH dpid: 2] Conectado con exito al controlador

```

DIFICULTADES ENCONTRADAS

Las mayores dificultades que encontramos en este trabajo práctico fueron en relación a las herramientas utilizadas.

Un primer problema que tuvimos fue lograr configurar un controlador por defecto a nuestra topología de Mininet para lo cual tuvimos que investigar bastante para lograr que Mininet reconozca a un controlador externo, en este caso POX.

En particular al utilizar POX, que cuenta con múltiples versiones, fuimos buscando información hasta dar con la adecuada para probar correctamente nuestra topología de Mininet con el controlador de POX. En nuestro caso, utilizamos la branch “fangtooth” de POX que utiliza Python 2. Con esa versión logramos que Mininet reconozca al controlador.

Luego al utilizar las herramientas surgieron otros problemas. Por ejemplo, a veces al intentar ejecutar el controlador surge que el puerto que se quiere utilizar ya está ocupado. Para solucionar esto, tuvimos que ejecutar el comando “*sudo fuser -k 6653/tcp*”, así liberamos el puerto y logramos arrancar POX.

Algo similar ocurre con Mininet. Por ejemplo, si se realiza un iperf sin éxito, quizás porque las reglas del firewall lo están bloqueando, Mininet se queda colgado y no hay otra opción que cerrarlo forzosamente. Luego algunos procesos de Mininet quedan corriendo en el background y no podemos iniciar Mininet nuevamente. Para solucionarlo, corrimos el comando “*sudo mn -c*”, limpiando estos procesos.

Otra de las dificultades con la que nos encontramos fue la escasa documentación para estas herramientas, en particular del controlador POX. A la hora de hacer funcionar adecuadamente el firewall en sí, iban surgiendo distintos problemas relacionados a la nomenclatura y los eventos de POX que no nos permitían ejecutar nuestras reglas del firewall. Por ejemplo, para aplicar nuestras reglas, descubrimos que siempre hay que aclarar el protocolo de transporte antes de restringir un puerto. Algo similar ocurre para elegir el switch que queremos que actúe como firewall, utilizando el `dpid`.