

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	2
2	Herramientas utilizadas	2
2.1	Mininet	2
2.2	POX	2
2.2.1	POX en ejecución	3
2.3	Wireshark & iperf	3
3	Resultados de simulaciones	5
3.1	Puerto Destino 80	5
3.1.1	Reglas	6
3.2	Iperf	6
3.2.1	Wireshark	6
3.2.2	Logs del controlador	6
3.3	Host 1, Puerto 5001 y UDP	8
3.3.1	Reglas	8
3.4	Iperf	8
3.4.1	Wireshark	9
3.4.2	Logs del controlador	9
3.5	Dos hosts no se comunican entre sí	10
3.5.1	Reglas	11
3.5.2	Wireshark	11
3.5.3	Logs del controlador	11
3.6	Mininet Pingall	12
4	Preguntas a responder	12
4.1	¿Cuál es la diferencia entre un Switch y un router? ¿Qué tienen en común?	12
4.2	¿Cuál es la diferencia entre un Switch convencional y un Switch OpenFlow?	13
4.3	¿Se pueden reemplazar todos los routers de la Internet por Switches OpenFlow? Piense en el escenario interASes para elaborar su respuesta.	13
5	Conclusión	13

1 Introducción

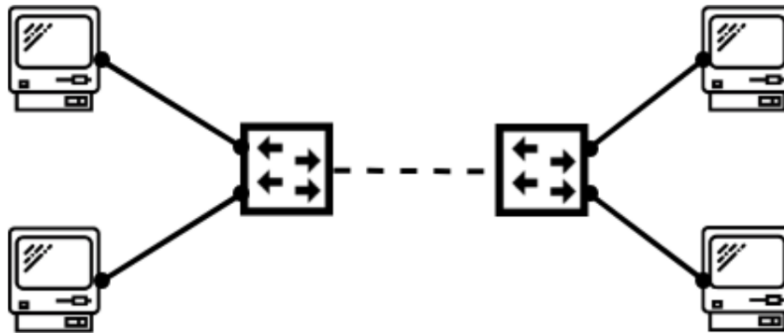
En este trabajo práctico se implementó un SDN que, mediante OpenFlow (utilizando POX), implementa un Firewall sobre una red creada en Mininet. Para ver el programa en acción y acercar la simulación a un caso de uso real, dentro de los hosts de la red de Mininet se utiliza iperf para establecer fácilmente 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

Para implementar el controlador con OpenFlow, se utilizó la biblioteca POX. El controlador utiliza P2 learning para que los switches aprendan automáticamente a reenviar paquetes.

Para el firewall implementamos el método `_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 utilización 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 ejecución

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

```
valen1611@ubuntu1611:~/code/redes$ make run
sed -i 's/IPDELSWITCHCONELFWALL = [0-9]/IPDELSWITCHCONELFWALL = 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 instalo

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.

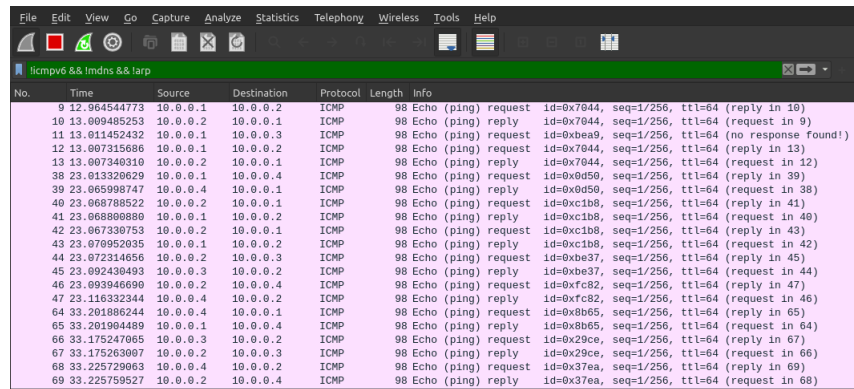
```

valen1611@ubuntu1611:~/code/redes$ sudo make mininet
mn --custom src/mytopo.py --topo mytopo,7 --controller remote,i
p=127.0.0.1,port=6633
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
mininet1 s2 s3 s4 s5 s6 s7
*** Adding links:
(h1, mininet1) (h2, mininet1) (h3, s7) (h4, s7) (mininet1, s2) (s
2, s3) (s3, s4) (s4, s5) (s5, s6) (s6, s7)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 7 switches
mininet1 s2 s3 s4 s5 s6 s7 ...
*** Starting Cl:
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)
mininet>

valen1611@ubuntu1611:~/code/redes$ make run
sed -i 's/IPDELSWITCHCONELFIREWALL = [0-9]/IPDELSWITCHCONELFIREWALL = 1/' src/fir
ewall.py
pox/pox.py firewall forwarding.l2_learning log.level --DEBUG samples.pretty_log
POX 0.6.0 (fangtooth) / Copyright 2011-2013 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] [00-00-00-00-00-02 5] connected
[forwarding.l2_learning] Connection [00-00-00-00-00-03 5]
[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] [00-00-00-00-00-05 7] connected
[forwarding.l2_learning] Connection [00-00-00-00-00-05 7]
[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] [00-00-00-00-00-07 2] connected
[forwarding.l2_learning] Connection [00-00-00-00-00-07 2]
[openflow.of_01] [00-00-00-00-00-01 1] connected
[firewall] Firewall rules installed on 00-00-00-00-00-01
[forwarding.l2_learning] Connection [00-00-00-00-00-01 1]
[openflow.of_01] [00-00-00-00-00-02 6] connected
[forwarding.l2_learning] Connection [00-00-00-00-00-02 6]
[forwarding.l2_learning] Installing flow for Sae2:f1:fa:d2:51.2 -> 16:95:7d:78:
e4:be.1
[forwarding.l2_learning] Installing flow for 16:95:7d:78:e4:be.1 -> Sae2:f1:fa:
d2:51.2
[forwarding.l2_learning] Installing flow for Sae2:f1:fa:d2:51.2 -> 16:95:7d:78:
e4:be.1
[forwarding.l2_learning] Installing flow for 2e:91:98:b2:56:33.2 -> 16:95:7d:78:
e4:be.1

```

Figure 2: Mostrando pox con Mininet a la vez



No.	Time	Source	Destination	Protocol	Length	Info
9	12.964544773	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x7044, seq=1/256, ttl=64 (reply in 10)
10	13.009485253	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x7044, seq=1/256, ttl=64 (request in 9)
11	13.011452432	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) request id=0x8ea9, seq=1/256, ttl=64 (no response found!)
12	13.007315686	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x7044, seq=1/256, ttl=64 (reply in 13)
13	13.007340310	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x7044, seq=1/256, ttl=64 (request in 12)
38	23.01320629	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) request id=0x0d50, seq=1/256, ttl=64 (reply in 39)
39	23.065998747	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0d50, seq=1/256, ttl=64 (request in 38)
40	23.068788522	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0xc1b8, seq=1/256, ttl=64 (reply in 41)
41	23.068800880	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0xc1b8, seq=1/256, ttl=64 (request in 40)
42	23.067330753	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0xc1b8, seq=1/256, ttl=64 (reply in 43)
43	23.070952035	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0xc1b8, seq=1/256, ttl=64 (request in 42)
44	23.072314656	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) request id=0xbe37, seq=1/256, ttl=64 (reply in 45)
45	23.092430493	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) reply id=0xbe37, seq=1/256, ttl=64 (request in 44)
46	23.093946690	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) request id=0xfc82, seq=1/256, ttl=64 (reply in 47)
47	23.116332344	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) reply id=0xfc82, seq=1/256, ttl=64 (request in 46)
64	33.201886244	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request id=0x8b65, seq=1/256, ttl=64 (reply in 65)
65	33.201904489	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply id=0x8b65, seq=1/256, ttl=64 (request in 64)
66	33.175247065	10.0.0.3	10.0.0.2	ICMP	98	Echo (ping) request id=0x29ce, seq=1/256, ttl=64 (reply in 67)
67	33.175263907	10.0.0.2	10.0.0.3	ICMP	98	Echo (ping) reply id=0x29ce, seq=1/256, ttl=64 (request in 66)
68	33.225729063	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) request id=0x37ea, seq=1/256, ttl=64 (reply in 69)
69	33.225759527	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) reply id=0x37ea, seq=1/256, ttl=64 (request in 68)

Figure 3: Wireshark con pingall

Para comprobar que no solo funciona con ICMP, utilizamos `iperf`. Con `iperf` simulamos clientes y servidores, usando conexiones tanto TCP como UDP. En el ejemplo a continuación, el host `h2` actúa como cliente y el host `h3` actúa como servidor, comunicándose por TCP.

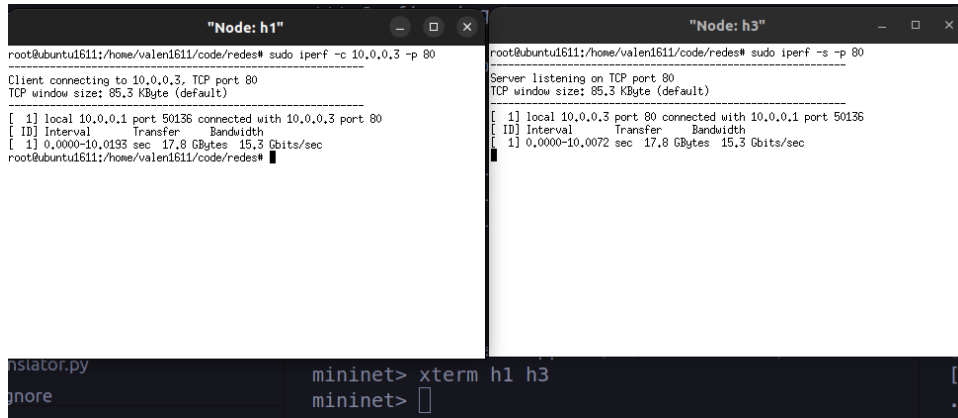


Figure 4: Hosts de Mininet con iperf básico

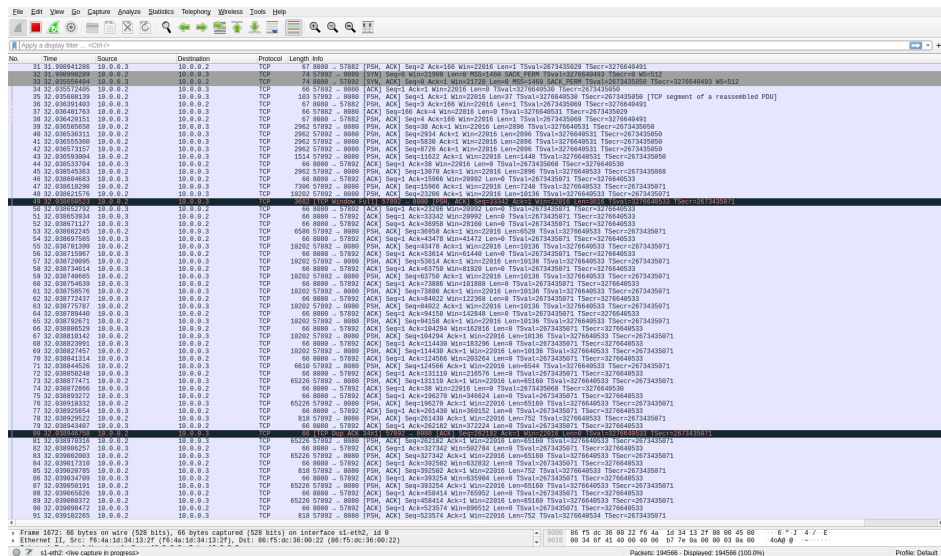


Figure 5: Imagen Traza Wireshark con el iperf básico

3 Resultados de simulaciones

3.1 Puerto Destino 80

Simulación para descartar todos los mensajes cuyo puerto destino sea 80. Se utilizan los hosts 1 y 4.

3.1.1 Reglas

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

3.2 Iperf

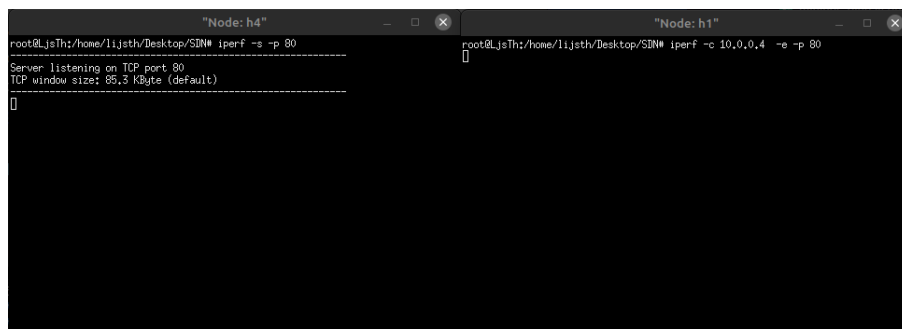


Figure 6: Iperf cliente y servidor TCP que no se pueden comunicar por el puerto 80

3.2.1 Wireshark

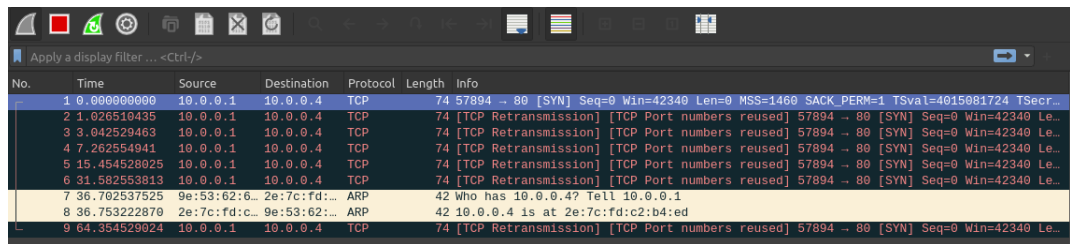


Figure 7: Wireshark con puerto 80 bloqueado mostrando que no llega el SYN

3.2.2 Logs del controlador

```
DEBUG:core:POX 0.6.0 (fangtooth) going up...
DEBUG:core:Running on CPython (2.7.18/Oct 15 2023 16:43:11)
DEBUG:core:Platform is Linux-5.15.0-112-generic-x86_64-with-LinuxMint-21-vanessa
INFO:core:POX 0.6.0 (fangtooth) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 2]
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
DEBUG:firewall:Firewall rules installed on 00-00-00-00-00-03
```

```

DEBUG:firewall:Installing policy: ofp_match
    dl_type: 0x800
    nw_proto: 6
    tp_dst: 80
DEBUG:firewall:Installing policy: ofp_match
    dl_type: 0x800
    nw_proto: 17
    tp_dst: 80
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-03 4]
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-02 5]
INFO:openflow.of_01:[00-00-00-00-00-04 3] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-04 3]
DEBUG:openflow.of_01:1 connection aborted
DEBUG:forwarding.l2_learning:installing flow for 2e:7c:fd:c2:b4:ed.3 -> 9e:53:62:66:03:ad.1
(...)
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.2
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
    dl_type: 0x800
    nw_proto: 6
    tp_dst: 80
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.3
(...)
DEBUG:forwarding.l2_learning:installing flow for 2e:7c:fd:c2:b4:ed.3 -> 9e:53:62:66:03:ad.1
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
    dl_type: 0x800
    nw_proto: 6
    tp_dst: 80
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.3
DEBUG:forwarding.l2_learning:installing flow for 2e:7c:fd:c2:b4:ed.3 -> 9e:53:62:66:03:ad.1
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
    dl_type: 0x800
    nw_proto: 6
    tp_dst: 80
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.3
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.2
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
    dl_type: 0x800
    nw_proto: 6
    tp_dst: 80
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.3
(...)
DEBUG:forwarding.l2_learning:installing flow for 2e:7c:fd:c2:b4:ed.3 -> 9e:53:62:66:03:ad.1
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
    dl_type: 0x800
    nw_proto: 6
    tp_dst: 80
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.3
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
    dl_type: 0x800
    nw_proto: 6
    tp_dst: 80
DEBUG:forwarding.l2_learning:installing flow for 9e:53:62:66:03:ad.1 -> 2e:7c:fd:c2:b4:ed.2

```

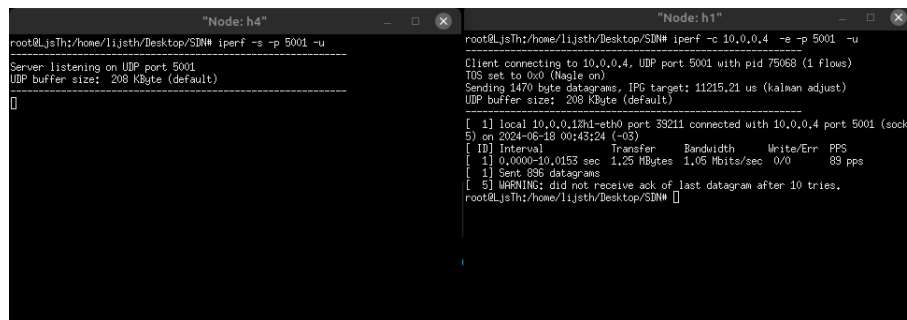

3.3 Host 1, Puerto 5001 y UDP

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

3.3.1 Reglas

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

3.4 Iperf



```
"Node: h4"
root@ljs1h:/home/lijsth/Desktop/SIN# iperf -s -p 5001 -u
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
[]

"Node: h1"
root@ljs1h:/home/lijsth/Desktop/SIN# iperf -c 10.0.0.4 -s -p 5001 -u
Client connecting to 10.0.0.4, UDP port 5001 with pid 75068 (1 flows)
TOS set to 0x0 (Nagle on)
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
[ 1] local 10.0.0.12h1-eth0 port 39211 connected with 10.0.0.4 port 5001 (sock=
5) on 2024-06-18 00:43:24 (-03)
[ 10] Interval      Transfer      Bandwidth    Write/Err    PPS
[ 1] 0.0000-10.0153 sec 1.25 MBytes  1.05 Mbits/sec  0/0      89 pps
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@ljs1h:/home/lijsth/Desktop/SIN# []
```

Figure 8: Iperf cliente en h1 y servidor en h4 UDP en puerto 5001 que no se pueden comunicar

3.4.1 Wireshark

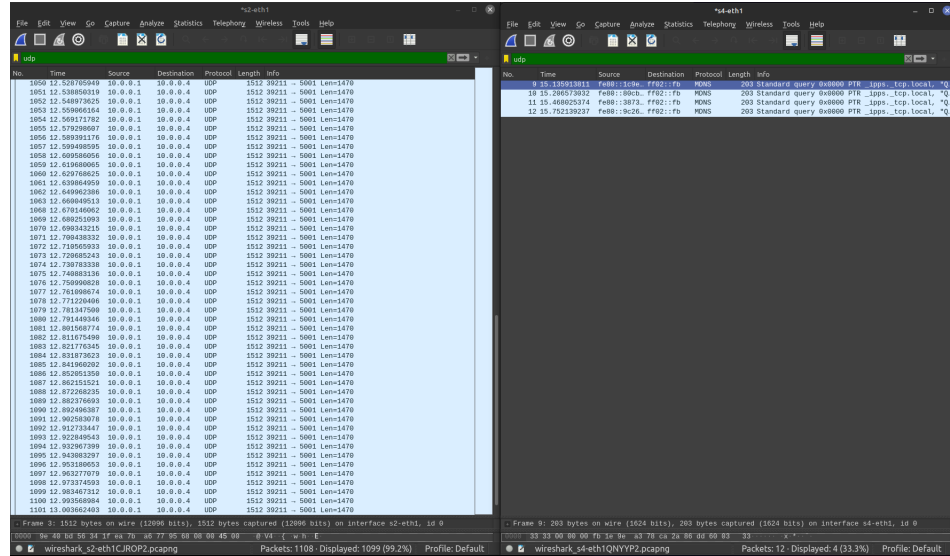


Figure 9: Logo de wireshark con el switch 2 (izquierda) y switch 4 (derecha) mostrando que los paquetes UDP de h1 no llegan a h4 luego después de pasar el firewall en switch 3

3.4.2 Logs del controlador

```
DEBUG:core:POX 0.6.0 (fangtooth) going up...
DEBUG:core:Running on CPython (2.7.18/Oct 15 2023 16:43:11)
DEBUG:core:Platform is Linux-5.15.0-112-generic-x86_64-with-LinuxMint-21-vanessa
INFO:core:POX 0.6.0 (fangtooth) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 2]
INFO:openflow.of_01:[00-00-00-00-00-03 4] connected
DEBUG:firewall:Firewall rules installed on 00-00-00-00-00-03
DEBUG:firewall:Installing policy: ofp_match
    dl_type: 0x800
    nw_proto: 17
    tp_dst: 80
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-03 4]
INFO:openflow.of_01:[00-00-00-00-00-02 5] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-02 5]
INFO:openflow.of_01:[00-00-00-00-00-04 3] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-04 3]
DEBUG:openflow.of_01:1 connection aborted
DEBUG:forwarding.l2_learning:installing flow for 9e:40:bd:56:34:1f.3 -> ea:7b:a6:77:95:68.1
DEBUG:forwarding.l2_learning:installing flow for 9e:40:bd:56:34:1f.2 -> ea:7b:a6:77:95:68.1
DEBUG:forwarding.l2_learning:installing flow for 9e:40:bd:56:34:1f.2 -> ea:7b:a6:77:95:68.1
DEBUG:forwarding.l2_learning:installing flow for 9e:40:bd:56:34:1f.3 -> ea:7b:a6:77:95:68.1
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
```

```

dl_type: 0x800
nw_proto: 17
nw_src: 10.0.0.1
tp_dst: 5001
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.3
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
dl_type: 0x800
nw_proto: 17
nw_src: 10.0.0.1
tp_dst: 5001
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.3
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
dl_type: 0x800
nw_proto: 17
nw_src: 10.0.0.1
tp_dst: 5001
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.3
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
dl_type: 0x800
nw_proto: 17
nw_src: 10.0.0.1
tp_dst: 5001
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.3
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.2
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
dl_type: 0x800
nw_proto: 17
nw_src: 10.0.0.1
tp_dst: 5001
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.3
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.2
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
dl_type: 0x800
nw_proto: 17
nw_src: 10.0.0.1
tp_dst: 5001
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.3
(...)
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.2
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.4 dropped with policy: ofp_match
dl_type: 0x800
nw_proto: 17
nw_src: 10.0.0.1
tp_dst: 5001
DEBUG:forwarding.l2_learning:installing flow for ea:7b:a6:77:95:68.1 -> 9e:40:bd:56:34:1f.3

```

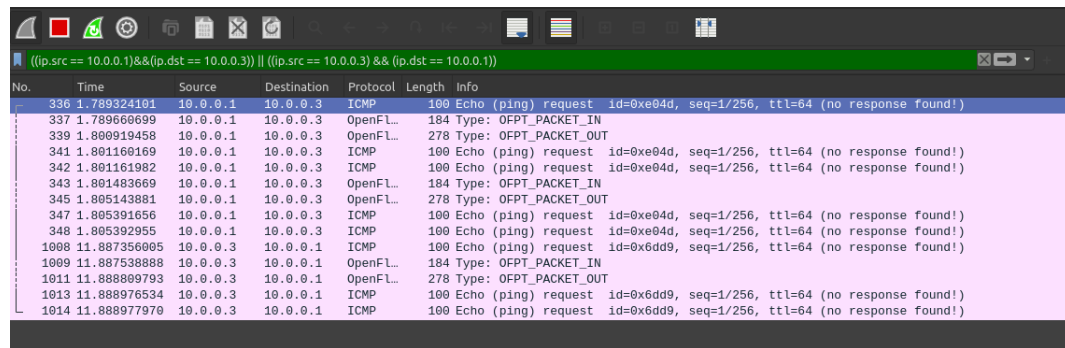
3.5 Dos hosts no se comunican entre sí

Simulación donde se eligen dos hosts cualquiera, y los mismos no pueden comunicarse de ninguna forma. Utilizamos el comando pingall de Mininet para demostrar que no se pueden comunicar. Se utilizan los hosts 1 y 3.

3.5.1 Reglas

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

3.5.2 Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
336	1.789324101	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) request id=0xe04d, seq=1/256, ttl=64 (no response found!)
337	1.789660699	10.0.0.1	10.0.0.3	OpenFL...	184	Type: OFPT_PACKET_IN
339	1.800919458	10.0.0.1	10.0.0.3	OpenFL...	278	Type: OFPT_PACKET_OUT
341	1.801160169	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) request id=0xe04d, seq=1/256, ttl=64 (no response found!)
342	1.801161982	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) request id=0xe04d, seq=1/256, ttl=64 (no response found!)
343	1.801483669	10.0.0.1	10.0.0.3	OpenFL...	184	Type: OFPT_PACKET_IN
345	1.805143881	10.0.0.1	10.0.0.3	OpenFL...	278	Type: OFPT_PACKET_OUT
347	1.805391656	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) request id=0xe04d, seq=1/256, ttl=64 (no response found!)
348	1.805392955	10.0.0.1	10.0.0.3	ICMP	100	Echo (ping) request id=0xe04d, seq=1/256, ttl=64 (no response found!)
1008	11.887356095	10.0.0.3	10.0.0.1	ICMP	100	Echo (ping) request id=0x6dd9, seq=1/256, ttl=64 (no response found!)
1009	11.887538888	10.0.0.3	10.0.0.1	OpenFL...	184	Type: OFPT_PACKET_IN
1011	11.888809793	10.0.0.3	10.0.0.1	OpenFL...	278	Type: OFPT_PACKET_OUT
1013	11.888976534	10.0.0.3	10.0.0.1	ICMP	100	Echo (ping) request id=0x6dd9, seq=1/256, ttl=64 (no response found!)
1014	11.888977970	10.0.0.3	10.0.0.1	ICMP	100	Echo (ping) request id=0x6dd9, seq=1/256, ttl=64 (no response found!)

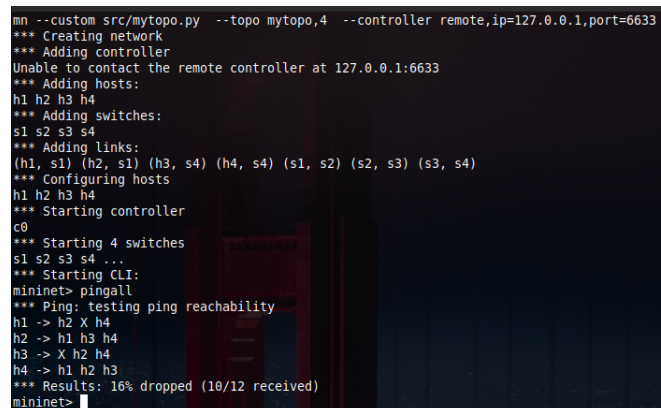
Figure 10: Wireshark con Banned Tuple

3.5.3 Logs del controlador

```
DEBUG:core:POX 0.6.0 (fangtooth) going up...
DEBUG:core:Running on CPython (2.7.18/Oct 15 2023 16:43:11)
DEBUG:core:Platform is Linux-5.15.0-112-generic-x86_64-with-LinuxMint-21-vanessa
INFO:core:POX 0.6.0 (fangtooth) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:firewall:Firewall rules installed on 00-00-00-00-00-03
DEBUG:firewall:Installing policy: ofp_match
  dl_type: 0x800
  nw_src: 10.0.0.1
  nw_dst: 10.0.0.3
DEBUG:firewall:Installing policy: ofp_match
  dl_type: 0x800
  nw_src: 10.0.0.3
  nw_dst: 10.0.0.1
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-03 3]
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 2]
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-02 4]
INFO:openflow.of_01:[00-00-00-00-00-04 1] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-04 1]
DEBUG:forwarding.l2_learning:installing flow for 42:42:63:8b:d0:39.2 -> 16:1f:83:b4:e5:53.1
```

```
(...)
DEBUG:forwarding.l2_learning:installing flow for a6:54:2f:87:1e:71.3 -> 16:1f:83:b4:e5:53.1
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.3 dropped with policy: ofp_match
    dl_type: 0x800
    nw_src: 10.0.0.1
    nw_dst: 10.0.0.3
DEBUG:forwarding.l2_learning:installing flow for 16:1f:83:b4:e5:53.1 -> a6:54:2f:87:1e:71.3
DEBUG:firewall:Packet from 10.0.0.1 -> 10.0.0.3 dropped with policy: ofp_match
    dl_type: 0x800
    nw_src: 10.0.0.1
    nw_dst: 10.0.0.3
DEBUG:forwarding.l2_learning:installing flow for 16:1f:83:b4:e5:53.1 -> 42:42:63:8b:d0:39.2
(...)
DEBUG:forwarding.l2_learning:installing flow for 86:01:c4:0d:11:a0.3 -> 42:42:63:8b:d0:39.2
DEBUG:firewall:Packet from 10.0.0.3 -> 10.0.0.1 dropped with policy: ofp_match
    dl_type: 0x800
    nw_src: 10.0.0.3
    nw_dst: 10.0.0.1
DEBUG:forwarding.l2_learning:installing flow for 86:01:c4:0d:11:a0.3 -> 16:1f:83:b4:e5:53.1
```

3.6 Mininet Pingall



```
mn --custom src/mytopo.py --topo mytopo,4 --controller remote,ip=127.0.0.1,port=6633
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6633
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1 s2 s3 s4
*** Adding links:
(h1, s1) (h2, s1) (h3, s4) (h4, s4) (s1, s2) (s2, s3) (s3, s4)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 4 switches
s1 s2 s3 s4
*** Starting CLI:
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 X h4
h2 -> h1 h3 h4
h3 -> X h2 h4
h4 -> h1 h2 h3
*** Results: 16% dropped (10/12 received)
mininet>
```

Figure 11: Mininet con Banned Tuple

4 Preguntas a responder

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

Un switch y un router son dispositivos de red que permiten la comunicación entre múltiples dispositivos, pero operan en diferentes capas. Un switch trabaja en la capa 2 (enlace) y utiliza direcciones MAC para redirigir paquetes, facilitando la comunicación entre dispositivos dentro de una misma red local. Por otro lado, un router opera en la capa 3 (red) y utiliza direcciones IP para

conectar múltiples redes simultáneamente, permitiendo la comunicación entre dispositivos en distintas redes. Ambos dispositivos toman los paquetes recibidos por sus puertos de entrada y los envían por los puertos de salida adecuados para alcanzar sus destinos finales.

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 integrar switches OpenFlow en una red, pero no es algo realmente factible debido a problemas de seguridad, rendimiento e interoperabilidad. En el contexto de interASes, manejar todo desde una única tecnología como OpenFlow podría facilitar la implementación de políticas de tráfico, pero crearía un single point of failure. Una vulnerabilidad en el protocolo OpenFlow expondría todo el Internet global. Además, sin routers, la implementación de NATs sería problemática. Los routers tradicionales, que implementan funciones en hardware, son más eficientes y rápidos, mientras que una red basada únicamente en switches OpenFlow estaría más congestionada. Además, los switches OpenFlow, como el POX, no implementan protocolos cruciales como BGP y OSPF, haciendo imposible la gestión completa de direcciones en una red global solo con switches OpenFlow.

5 Conclusión

Gracias a la tecnología de Openflow; pudimos codificar un firewall dinámico. Este nos permitió describir un set de reglas para filtrar paquetes y luego instalar dichas reglas de forma remota a un switch.