

Software Defined Networks

Materia: Introducción a los Sistemas
Distribuidos (75.43) - Redes (TA048)

Alumnos:

Ascencio Felipe Santino 110675

Burgos Moreno Daniel 110486

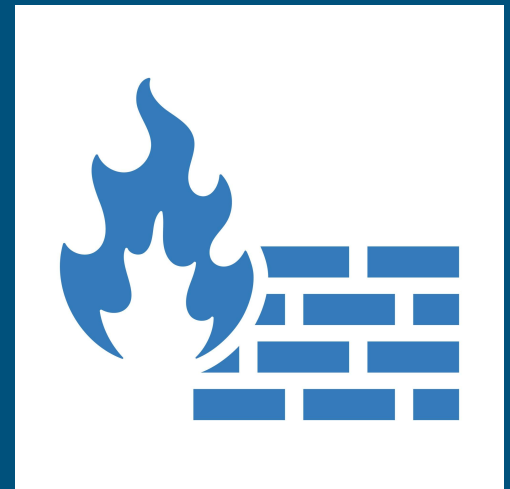
García Pizales Ignacio 105043

Levi Dolores 105993

Orive María Sol 91351

Hipotesis

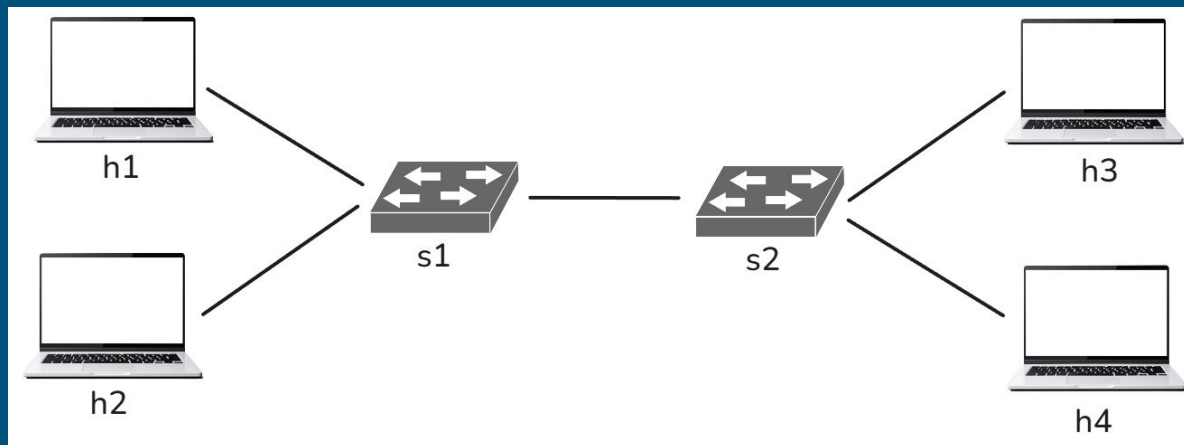
- Implementamos un firewall definido por software utilizando POX + Mininet.
- Las reglas del firewall se leen desde un archivo .json.
- Se bloquean flujos específicos según IP, puerto, MAC y protocolo.



Topología

Cadena de switches + 4 hosts

- Hosts: h1, h2, h3, h4.
- Switches conectados en serie.
- Reglas aplicadas solo en dpid = 2.



Controlador POX

- Escucha eventos ConnectionUp.
- Aplica reglas con `ofp_match()` y `ofp_flow_mod()`.
- Compatible con múltiples switches.
- Archivo JSON.
- Externo al código.
- Permite modificar reglas sin recompilar.



Reglas utilizadas

- Bloquear todo tráfico TCP con destino a puerto 80.
- Bloquear tráfico UDP al puerto 5001 desde h1.
- Bloquear toda comunicación entre h1 y h4 (ida y vuelta).

```
[
  {
    "switches": [2],
    "tipo_protocolo_transporte": "tcp",
    "puerto_destino": 80
  },
  {
    "switches": [2],
    "tipo_protocolo_transporte": "udp",
    "puerto_destino": 80
  },
  {
    "switches": [2],
    "tipo_protocolo_transporte": "udp",
    "puerto_destino": 5001,
    "mac_origen": "00:00:00:00:00:01"
  },
  {
    "switches": [2],
    "mac_origen": "00:00:00:00:00:01",
    "mac_destino": "00:00:00:00:00:04"
  },
  {
    "switches": [2],
    "mac_origen": "00:00:00:00:00:04",
    "mac_destino": "00:00:00:00:00:01"
  }
]
```

Pasos principales

- Ejecutar controlador POX - 'controlador.sh'.
- Levantar Mininet con topología personalizada - 'topologia_cadena.sh'.
- Enviar tráfico con iperf.
- De ser deseado un análisis más profundo: Capturar tráfico con Wireshark.

Podemos ver todo este paso a paso documentado en ambos informes.

Ejemplo de ejecución inicial del 'Firewall'

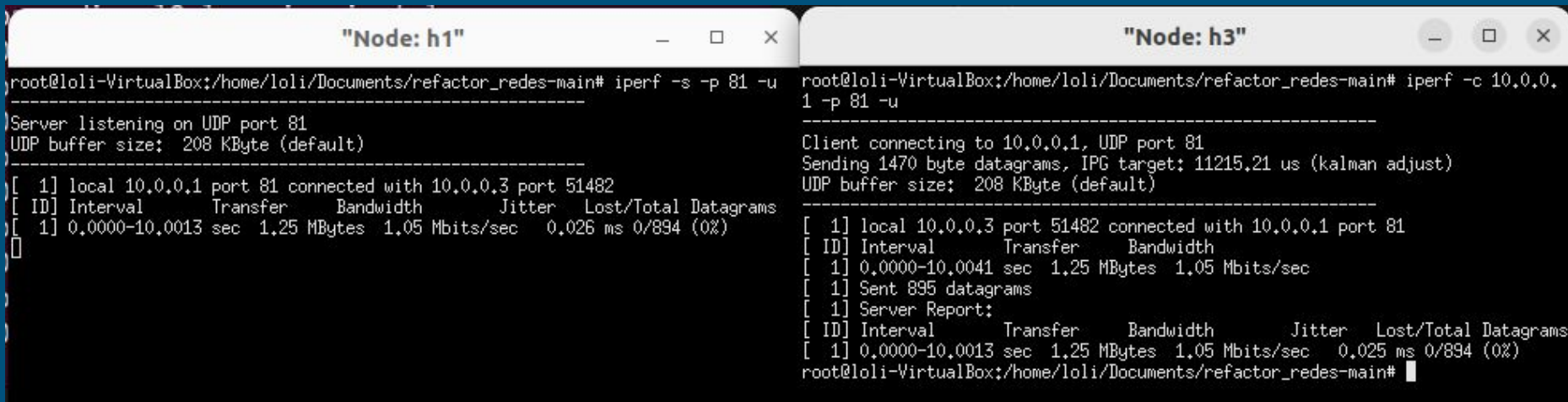
```
loli@loli-VirtualBox:~/Documents/refactor_redes-main$ ./controlador.sh
POX 0.7.0 (gar) / Copyright 2011-2020 James McCauley, et al.
reglas.json
DEBUG:controller:Habilitando el Firewall.
DEBUG:core:POX 0.7.0 (gar) going up...
DEBUG:core:Running on CPython (3.10.12/Feb 4 2025 14:57:36)
DEBUG:core:Platform is Linux-6.8.0-59-generic-x86_64-with-glibc2.35
WARNING:version:POX requires one of the following versions of Python: 3.6 3.7 3.8 3.9
WARNING:version:You're running Python 3.10.
WARNING:version:If you run into problems, try using a supported version.
INFO:core:POX 0.7.0 (gar) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
DEBUG:openflow.of_01:[00-00-00-00-00-02 4] Got early port status message for port 65534
INFO:openflow.of_01:[00-00-00-00-00-02 4] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-02 4]
DEBUG:controller:Reglas del Firewall instaladas en 00-00-00-00-00-02
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-01 2]
DEBUG:controller:Reglas del Firewall instaladas en 00-00-00-00-00-01
INFO:openflow.of_01:[00-00-00-00-00-03 3] connected
DEBUG:forwarding.l2_learning:Connection [00-00-00-00-00-03 3]
DEBUG:controller:Reglas del Firewall instaladas en 00-00-00-00-00-03
```

Ejemplo de ejecución inicial de 'Mininet'

```
loli@loli-VirtualBox:~/Documents/refactor_redes-main$ ./topologia_cadena.sh 3
*** Removing excess controllers/ofprotocols/ofdatapaths/pings/noxes
killall controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
killall -9 controller ofprotocol ofdatapath ping nox_corelt-nox_core ovs-openflowd ovs-controllerovs-testcontroller udpbwtest mnexec ivs ryu-manager 2> /dev/null
pkill -9 -f "sudo mnexec"
*** Removing junk from /tmp
rm -f /tmp/vconn* /tmp/vlogs* /tmp/*.out /tmp/*.log
*** Removing old X11 tunnels
*** Removing excess kernel datapaths
ps ax | egrep -o 'dp[0-9]+' | sed 's/dp/nl:/'
*** Removing OVS datapaths
ovs-vsctl --timeout=1 list-br
ovs-vsctl --timeout=1 list-br
*** Removing all links of the pattern foo-ethX
ip link show | egrep -o '([_-[:alnum:]]+-eth[[:digit:]]+)'
ip link show
*** Killing stale mininet node processes
pkill -9 -f mininet:
*** Shutting down stale tunnels
pkill -9 -f Tunnel=Ethernet
pkill -9 -f .ssh/mn
rm -f ~/.ssh/mn/*
*** Cleanup complete.
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6653
Setting remote controller to 127.0.0.1:6653
*** 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> 9
```


Muestra de funcionamiento con la 'Regla 1'

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



The image shows two terminal windows side-by-side, both titled "Node: h1" and "Node: h3". The left window (Node: h1) shows the output of the command `iperf -s -p 81 -u`, indicating it is listening on UDP port 81. It shows a connection from 10.0.0.3 port 51482 and reports a bandwidth of 1.05 Mbits/sec. The right window (Node: h3) shows the output of the command `iperf -c 10.0.0.1 -p 81 -u`, indicating it is connecting to 10.0.0.1 on UDP port 81. It shows sending 1470 byte datagrams and reports a bandwidth of 1.05 Mbits/sec. Both windows show a loss of 0/894 (0%).

```
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 81 -u
Server listening on UDP port 81
UDP buffer size: 208 KByte (default)

[ 1] local 10.0.0.1 port 81 connected with 10.0.0.3 port 51482
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[ 1] 0.0000-10.0013 sec 1.25 MBytes 1.05 Mbits/sec 0.026 ms 0/894 (0%)

root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.1 -p 81 -u
Client connecting to 10.0.0.1, UDP port 81
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)

[ 1] local 10.0.0.3 port 51482 connected with 10.0.0.1 port 81
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[ 1] 0.0000-10.0041 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 895 datagrams
[ 1] Server Report:
[ ID] Interval      Transfer    Bandwidth      Jitter    Lost/Total Datagrams
[ 1] 0.0000-10.0013 sec 1.25 MBytes 1.05 Mbits/sec 0.025 ms 0/894 (0%)
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#
```

Muestra de funcionamiento con la 'Regla 2'

```
"Node: h1"                                     "Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 80      root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.
Server listening on TCP port 80
TCP window size: 85.3 KByte (default)
1 -p 80
```

```
"Node: h1"                                     "Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 80 -u  root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.
Server listening on UDP port 80
UDP buffer size: 208 KByte (default)
1 -p 80 -u

Client connecting to 10.0.0.1, UDP port 80
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)

[ 1] local 10.0.0.3 port 37320 connected with 10.0.0.1 port 80
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0166 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#
```

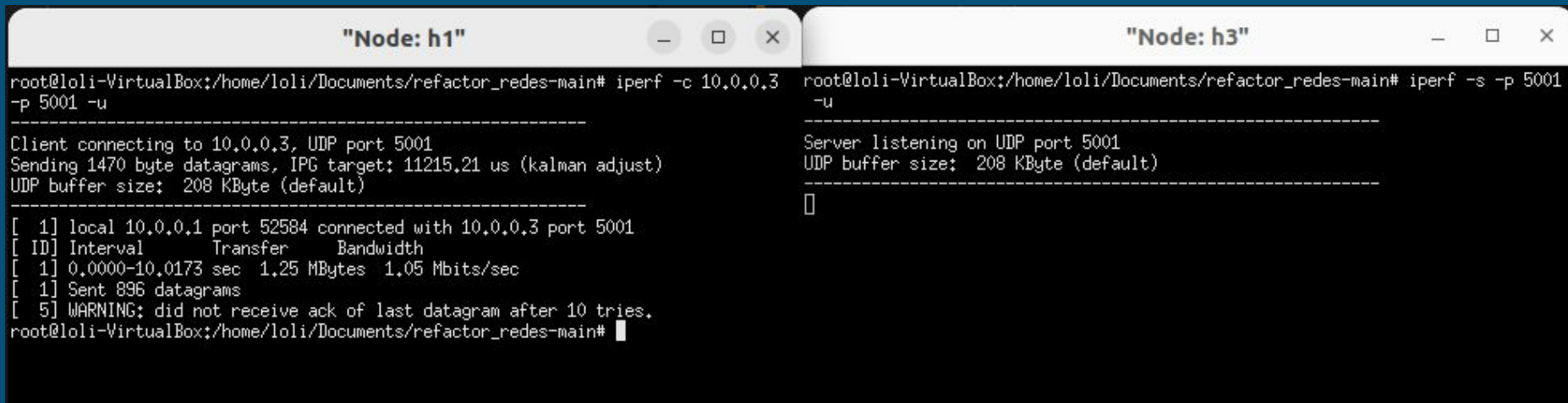
```
"Node: h1"                                     "Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 81      root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.
Server listening on TCP port 81
TCP window size: 85.3 KByte (default)
1 -p 81

[ 1] local 10.0.0.1 port 81 connected with 10.0.0.3 port 41366
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-9.9976 sec 27.3 GBytes 23.4 Gbits/sec
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#

Client connecting to 10.0.0.1, TCP port 81
TCP window size: 85.3 KByte (default)

[ 1] local 10.0.0.3 port 41366 connected with 10.0.0.1 port 81
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0145 sec 27.3 GBytes 23.4 Gbits/sec
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#
```

Muestra de funcionamiento con la 'Regla 3'



The image displays two terminal windows side-by-side, representing a network test setup. The left window, titled "Node: h1", shows the execution of the iperf client command. It reports connecting to 10.0.0.3 on port 5001, sending 1470 byte datagrams, and achieving a bandwidth of 1.05 Mbits/sec. The right window, titled "Node: h3", shows the iperf server listening on port 5001. Both windows show the standard iperf output format with a table of results.

```
"Node: h1"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -c 10.0.0.3 -p 5001 -u
-----
Client connecting to 10.0.0.3, UDP port 5001
Sending 1470 byte datagrams, IPG target: 11215.21 us (kalman adjust)
UDP buffer size: 208 KByte (default)
-----
[ 1] local 10.0.0.1 port 52584 connected with 10.0.0.3 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 1] 0.0000-10.0173 sec 1.25 MBytes 1.05 Mbits/sec
[ 1] Sent 896 datagrams
[ 5] WARNING: did not receive ack of last datagram after 10 tries.
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main#
```

```
"Node: h3"
root@loli-VirtualBox:/home/loli/Documents/refactor_redes-main# iperf -s -p 5001 -u
-----
Server listening on UDP port 5001
UDP buffer size: 208 KByte (default)
-----
[]
```

Dificultades en el Desarrollo

- Mininet requiere tiempo para entender su lógica y perfeccionar su manejo.
- Sintaxis de reglas POX en JSON es sensible.
- Dependencias de POX.
- Interpretación de flujos a bajo nivel en Wireshark.

Conclusión

POX + Mininet permiten controlar tráfico de red en detalle

- Programar reglas desde software aporta gran flexibilidad.
- Las SDN requieren mayor esfuerzo inicial, pero son más potentes.
- El TP nos ayudó a entender cómo se gestionan flujos y se aplican políticas dinámicas en red.