Redes Software - Desarrollo SDN

Grupo 2
100451366 - Noel Andolz Aguado
100429717 - David Peño Gutiérrez

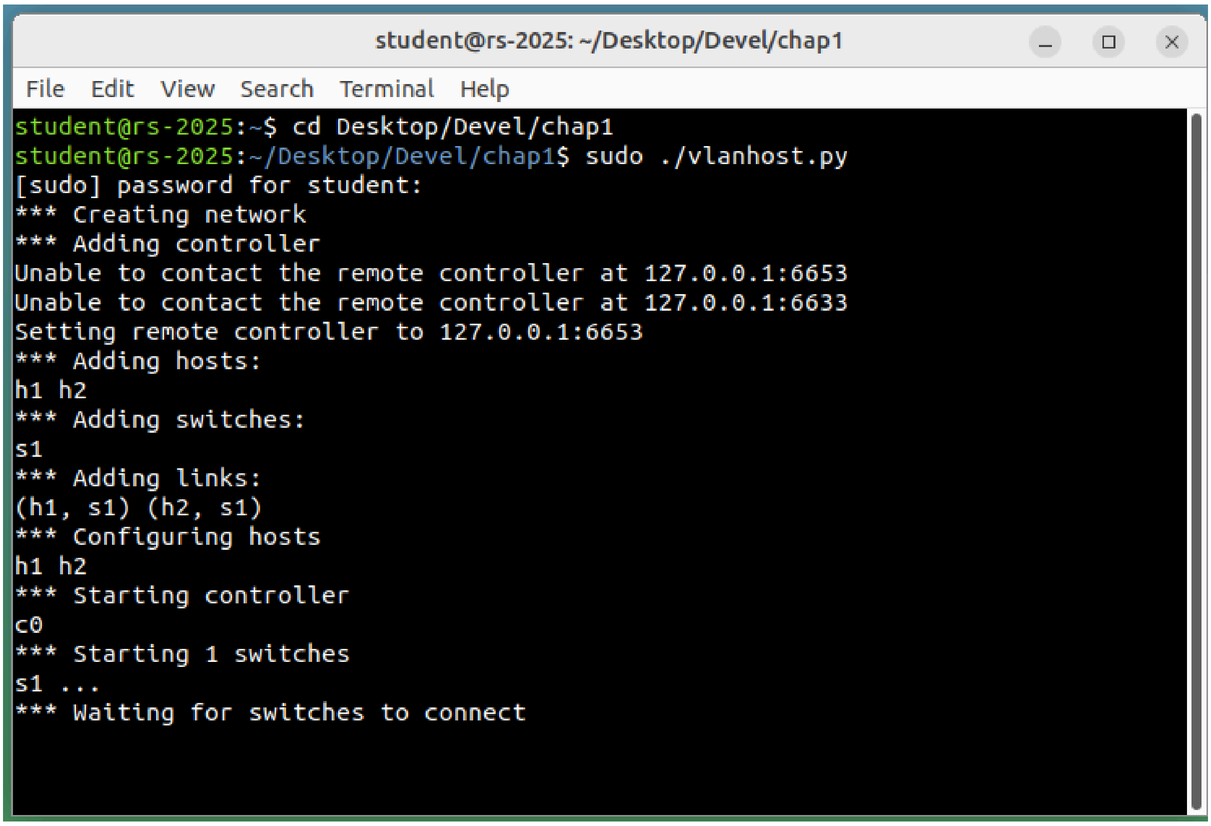
Copiar y pegar los ficheros de código fuente y poner fuente `Roboto Mono Normal - 10pt`

Hito 1:

- scenario.py → *vlanhost.py*
- simple_router.py → *L3Switch.py*
- Una captura de pantalla del terminal mininet con el comando `h1 ping -c4 h2`

*Los códigos de *vlanhost.py* y *L3Switch.py* se encuentran al final del hito

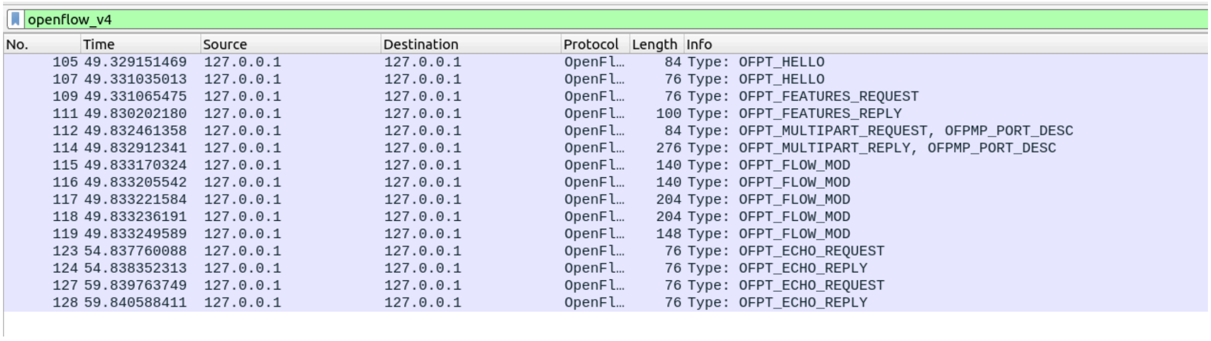
Primero ejecutamos mininet en un terminal.
Una vez vemos lo siguiente:



student@rs-2025: ~/Desktop/Devel/chap1
File Edit View Search Terminal Help

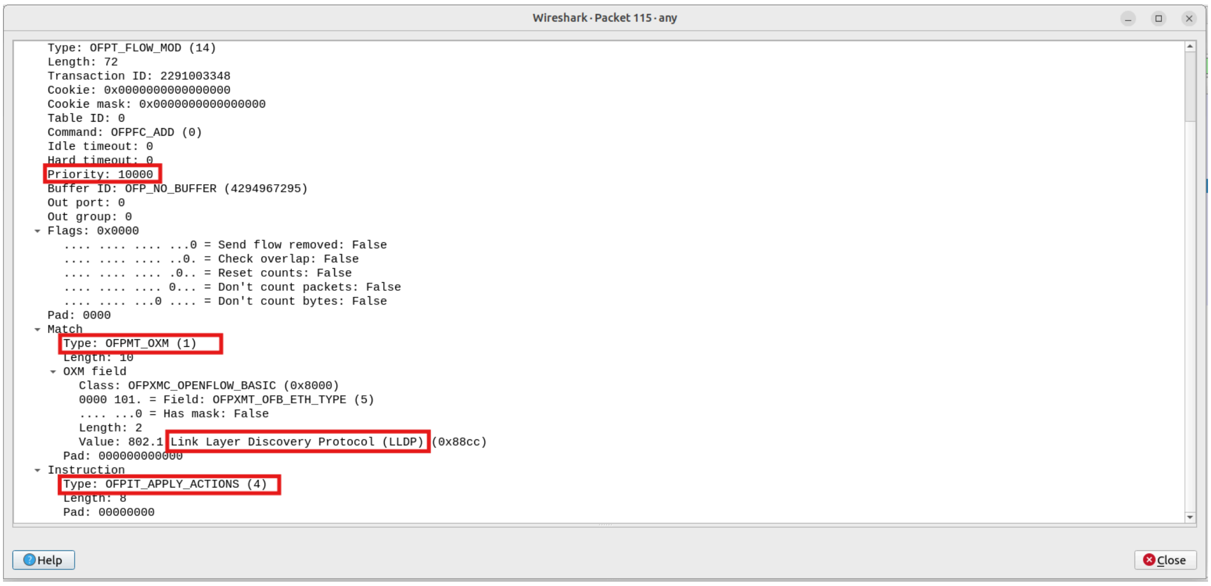
```
student@rs-2025:~$ cd Desktop/Devel/chap1
student@rs-2025:~/Desktop/Devel/chap1$ sudo ./vlanhost.py
[sudo] password for student:
*** Creating network
*** Adding controller
Unable to contact the remote controller at 127.0.0.1:6653
Unable to contact the remote controller at 127.0.0.1:6633
Setting remote controller to 127.0.0.1:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
```

Abrimos Wireshark para poder ver la interacción entre el controlador y la topología de mininet.
Elegimos *any* para ver todo el tráfico. Y aplicamos el filtro “*openflow_v4*”.
Vemos lo siguiente:



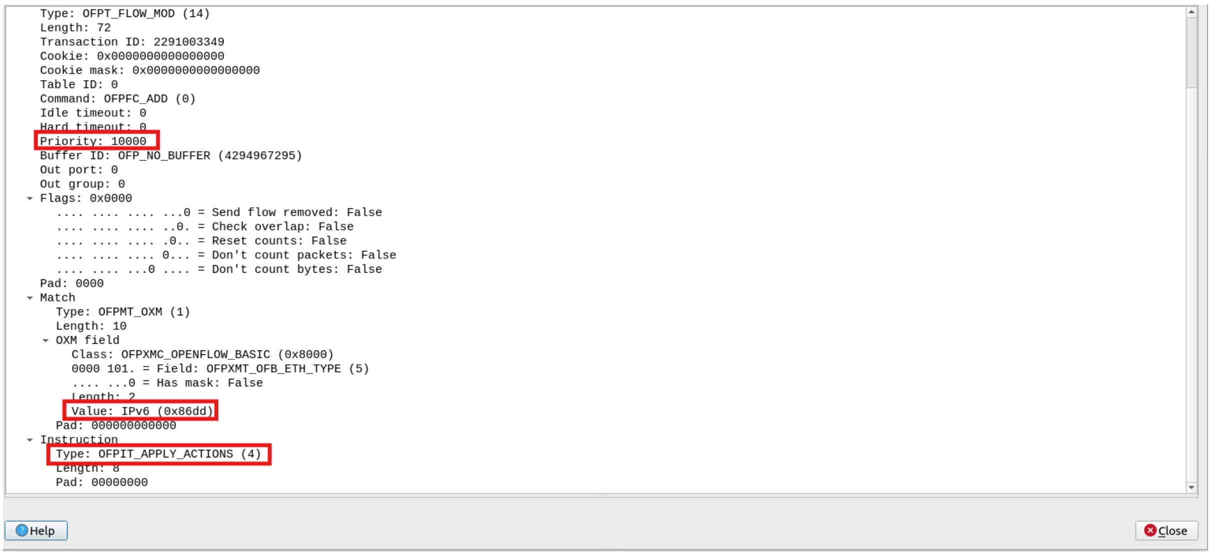
```
No.    Time          Source       Destination   Protocol  Length Info
   105 49.329151469  127.0.0.1    127.0.0.1     OpenFl…      84 Type: OFPT_HELLO
   107 49.331035013  127.0.0.1    127.0.0.1     OpenFl…      76 Type: OFPT_HELLO
   109 49.331065475  127.0.0.1    127.0.0.1     OpenFl…      76 Type: OFPT_FEATURES_REQUEST
   111 49.830202180  127.0.0.1    127.0.0.1     OpenFl…     100 Type: OFPT_FEATURES_REPLY
   112 49.832461358  127.0.0.1    127.0.0.1     OpenFl…      84 Type: OFPT_MULTIPART_REQUEST, OFPMP_PORT_DESC
   114 49.832912341  127.0.0.1    127.0.0.1     OpenFl…     276 Type: OFPT_MULTIPART_REPLY, OFPMP_PORT_DESC
   115 49.833170324  127.0.0.1    127.0.0.1     OpenFl…     140 Type: OFPT_FLOW_MOD
   116 49.833205542  127.0.0.1    127.0.0.1     OpenFl…     140 Type: OFPT_FLOW_MOD
   117 49.833221584  127.0.0.1    127.0.0.1     OpenFl…     204 Type: OFPT_FLOW_MOD
   118 49.833236191  127.0.0.1    127.0.0.1     OpenFl…     204 Type: OFPT_FLOW_MOD
   119 49.833249589  127.0.0.1    127.0.0.1     OpenFl…     148 Type: OFPT_FLOW_MOD
   123 54.837760088  127.0.0.1    127.0.0.1     OpenFl…      76 Type: OFPT_ECHO_REQUEST
   124 54.838352313  127.0.0.1    127.0.0.1     OpenFl…      76 Type: OFPT_ECHO_REPLY
   127 59.839763749  127.0.0.1    127.0.0.1     OpenFl…      76 Type: OFPT_ECHO_REQUEST
   128 59.840588411  127.0.0.1    127.0.0.1     OpenFl…      76 Type: OFPT_ECHO_REPLY
```

Podemos ver que en los dos primeros mensajes capturados, se está iniciando la conexión.
Después, vemos tanto el “*features request*” como el “*features reply*”.
Después, vemos una serie de mensajes en los que se envían las reglas:



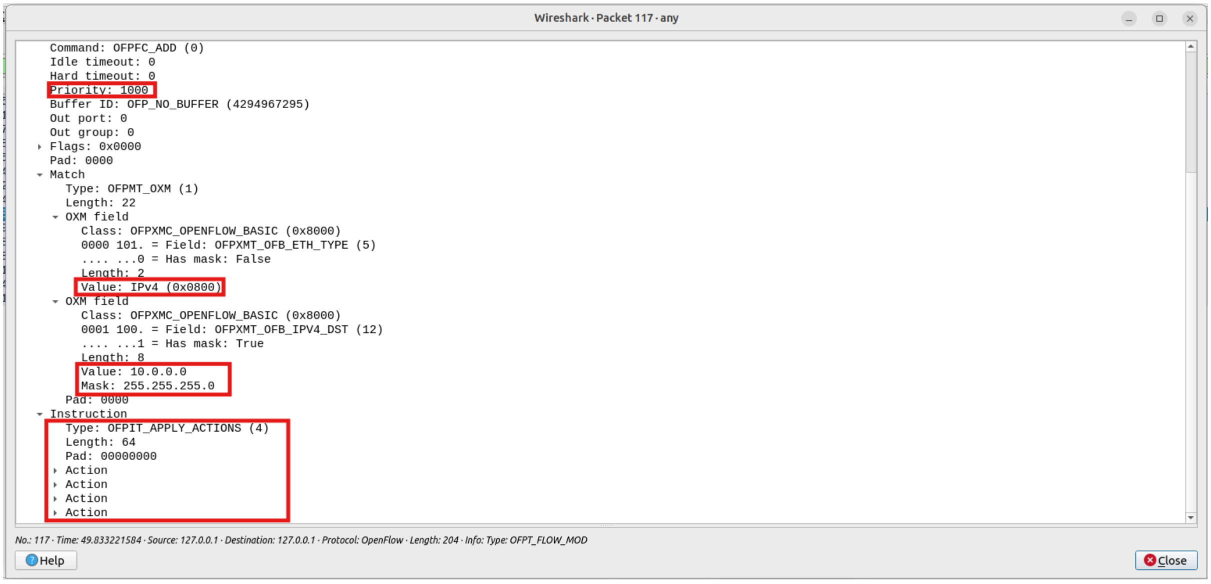
```
    Type: OFPT_FLOW_MOD (14)
    Length: 72
    Transaction ID: 2291003348
    Cookie: 0x0000000000000000
    Cookie mask: 0x0000000000000000
    Table ID: 0
    Command: OFPFC_ADD (0)
    Idle timeout: 0
    Hard timeout: 0
    Priority: 10000
    Buffer ID: OFP_NO_BUFFER (4294967295)
    Out port: 0
    Out group: 0
  ▾ Flags: 0x0000
      .... .... .... ...0 = Send flow removed: False
      .... .... .... ..0. = Check overlap: False
      .... .... .... .0.. = Reset counts: False
      .... .... .... 0... = Don't count packets: False
      .... .... ...0 .... = Don't count bytes: False
    Pad: 0000
  ▾ Match
      Type: OFPMT_OXM (1)
      Length: 10
    ▾ OXM field
        Class: OFPXMC_OPENFLOW_BASIC (0x8000)
        0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
        .... ...0 = Has mask: False
        Length: 2
        Value: 802.1 Link Layer Discovery Protocol (LLDP) (0x88cc)
      Pad: 000000000000
  ▾ Instruction
      Type: OFPIT_APPLY_ACTIONS (4)
      Length: 8
      Pad: 00000000
```

En esta primera, vemos las instrucciones para el tráfico LLDP, que tiene una prioridad de 10000 y que se descarta.



```
    Type: OFPT_FLOW_MOD (14)
    Length: 72
    Transaction ID: 2291003349
    Cookie: 0x0000000000000000
    Cookie mask: 0x0000000000000000
    Table ID: 0
    Command: OFPFC_ADD (0)
    Idle timeout: 0
    Hard timeout: 0
    Priority: 10000
    Buffer ID: OFP_NO_BUFFER (4294967295)
    Out port: 0
    Out group: 0
  ▾ Flags: 0x0000
      .... .... .... ...0 = Send flow removed: False
      .... .... .... ..0. = Check overlap: False
      .... .... .... .0.. = Reset counts: False
      .... .... .... 0... = Don't count packets: False
      .... .... ...0 .... = Don't count bytes: False
    Pad: 0000
  ▾ Match
      Type: OFPMT_OXM (1)
      Length: 10
    ▾ OXM field
        Class: OFPXMC_OPENFLOW_BASIC (0x8000)
        0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
        .... ...0 = Has mask: False
        Length: 2
        Value: IPv6 (0x86dd)
      Pad: 000000000000
  ▾ Instruction
      Type: OFPIT_APPLY_ACTIONS (4)
      Length: 8
      Pad: 00000000
```

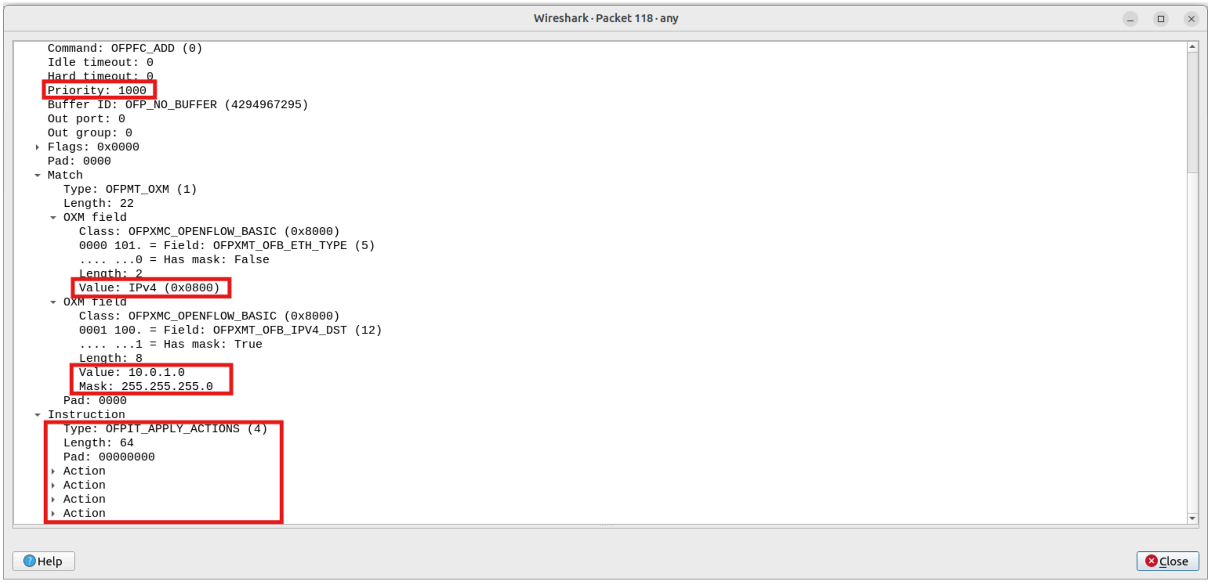
En este mensaje, podemos ver que se envían las instrucciones para el tráfico IPv6, que de nuevo tiene prioridad 10000 y también se descarta.



```
    Command: OFPFC_ADD (0)
    Idle timeout: 0
    Hard timeout: 0
    Priority: 1000
    Buffer ID: OFP_NO_BUFFER (4294967295)
    Out port: 0
    Out group: 0
  ▸ Flags: 0x0000
    Pad: 0000
  ▾ Match
      Type: OFPMT_OXM (1)
      Length: 22
    ▾ OXM field
        Class: OFPXMC_OPENFLOW_BASIC (0x8000)
        0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
        .... ...0 = Has mask: False
        Length: 2
        Value: IPv4 (0x0800)
    ▾ OXM field
        Class: OFPXMC_OPENFLOW_BASIC (0x8000)
        0001 100. = Field: OFPXMT_OFB_IPV4_DST (12)
        .... ...1 = Has mask: True
        Length: 8
        Value: 10.0.0.0
        Mask: 255.255.255.0
      Pad: 0000
  ▾ Instruction
      Type: OFPIT_APPLY_ACTIONS (4)
      Length: 64
      Pad: 00000000
    ▸ Action
    ▸ Action
    ▸ Action
    ▸ Action
```

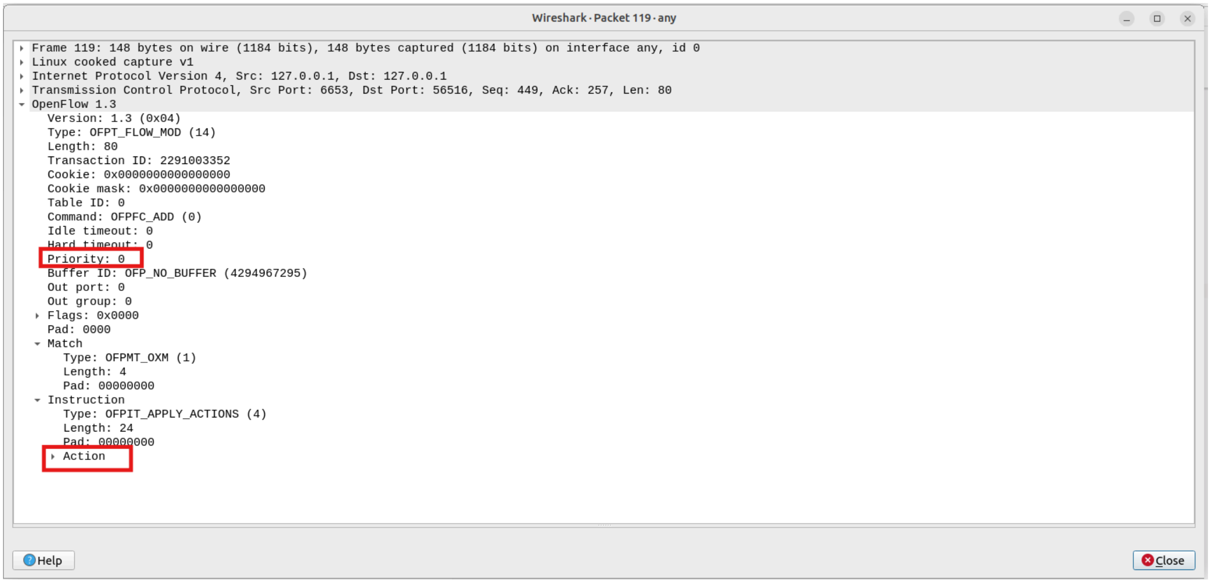
No.: 117 · Time: 49.833221584 · Source: 127.0.0.1 · Destination: 127.0.0.1 · Protocol: OpenFlow · Length: 204 · Info: Type: OFPT_FLOW_MOD

En este mensaje, vemos que se están enviando las reglas para el tráfico IPv4 cuyo prefijo es 10.0.0.0/24.



```
    Command: OFPFC_ADD (0)
    Idle timeout: 0
    Hard timeout: 0
    Priority: 1000
    Buffer ID: OFP_NO_BUFFER (4294967295)
    Out port: 0
    Out group: 0
  ▸ Flags: 0x0000
    Pad: 0000
  ▾ Match
      Type: OFPMT_OXM (1)
      Length: 22
    ▾ OXM field
        Class: OFPXMC_OPENFLOW_BASIC (0x8000)
        0000 101. = Field: OFPXMT_OFB_ETH_TYPE (5)
        .... ...0 = Has mask: False
        Length: 2
        Value: IPv4 (0x0800)
    ▾ OXM field
        Class: OFPXMC_OPENFLOW_BASIC (0x8000)
        0001 100. = Field: OFPXMT_OFB_IPV4_DST (12)
        .... ...1 = Has mask: True
        Length: 8
        Value: 10.0.1.0
        Mask: 255.255.255.0
      Pad: 0000
  ▾ Instruction
      Type: OFPIT_APPLY_ACTIONS (4)
      Length: 64
      Pad: 00000000
    ▸ Action
    ▸ Action
    ▸ Action
    ▸ Action
```

En este mensaje, vemos las reglas que se envían en el caso de tener tráfico IPv4 y cuyo prefijo sea 10.0.1.0/24.



```
▸ Frame 119: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface any, id 0
▸ Linux cooked capture v1
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 6653, Dst Port: 56516, Seq: 449, Ack: 257, Len: 80
▾ OpenFlow 1.3
    Version: 1.3 (0x04)
    Type: OFPT_FLOW_MOD (14)
    Length: 80
    Transaction ID: 2291003352
    Cookie: 0x0000000000000000
    Cookie mask: 0x0000000000000000
    Table ID: 0
    Command: OFPFC_ADD (0)
    Idle timeout: 0
    Hard timeout: 0
    Priority: 0
    Buffer ID: OFP_NO_BUFFER (4294967295)
    Out port: 0
    Out group: 0
  ▸ Flags: 0x0000
    Pad: 0000
  ▾ Match
      Type: OFPMT_OXM (1)
      Length: 4
      Pad: 00000000
  ▾ Instruction
      Type: OFPIT_APPLY_ACTIONS (4)
      Length: 24
      Pad: 00000000
    ▸ Action
```

Finalmente, tenemos el mensaje en el que se transmite la regla por defecto, indicando que se envía al controlador y con una prioridad de 0.

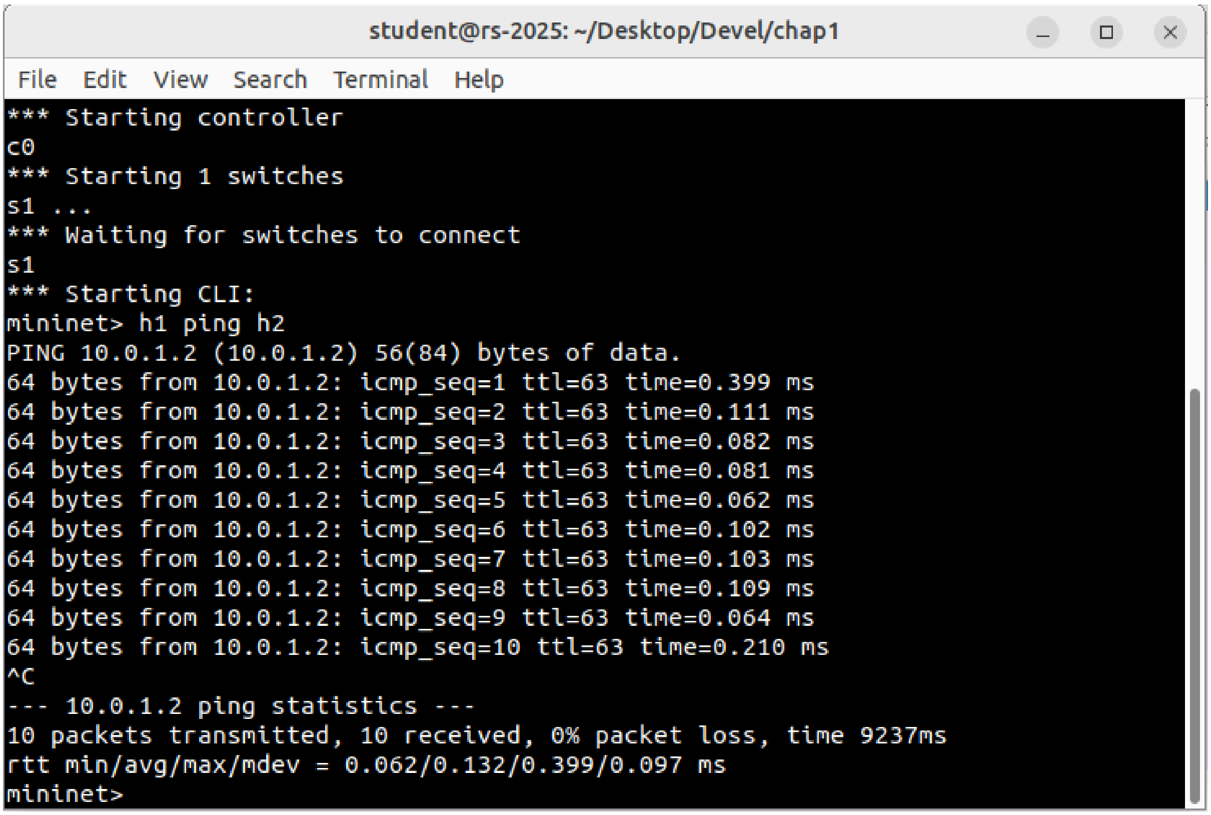
El siguiente tráfico que se puede ver son mensajes "*echo request*" y "*echo reply*" para asegurarse de que la conexión sigue establecida. Además, aprovechan para comprobar la latencia,...

Una vez vemos que el intercambio ha sido exitoso, podemos cerrar todo.

Volvemos a iniciar la topología, el controlador y Wireshark.

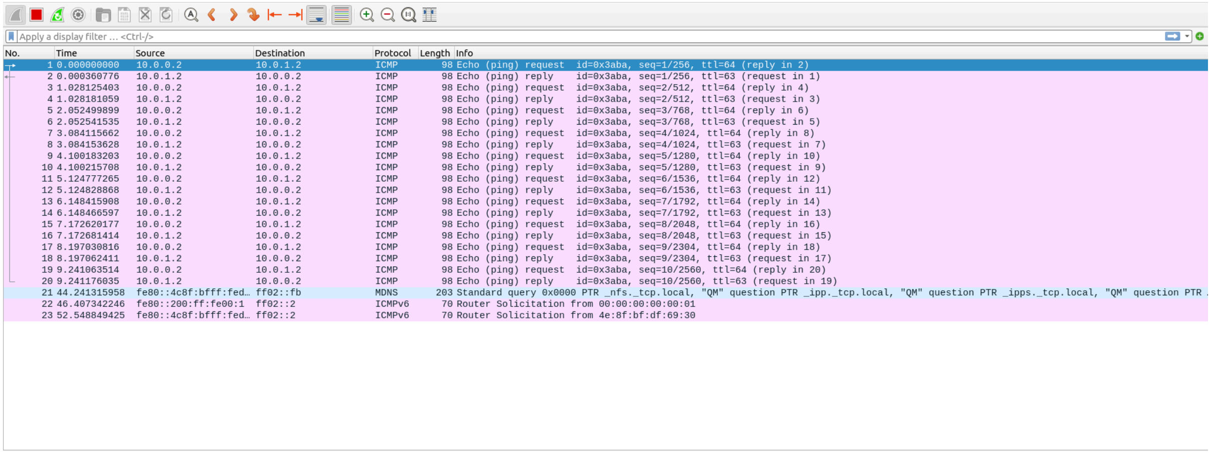
En Wireshark, en vez de capturar el tráfico en *any*, lo capturamos en *s1-eth1*.

Hacemos un ping desde h1 hasta h2:



```
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Waiting for switches to connect
s1
*** Starting CLI:
mininet> h1 ping h2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=63 time=0.399 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=63 time=0.111 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=63 time=0.082 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=63 time=0.081 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=63 time=0.062 ms
64 bytes from 10.0.1.2: icmp_seq=6 ttl=63 time=0.102 ms
64 bytes from 10.0.1.2: icmp_seq=7 ttl=63 time=0.103 ms
64 bytes from 10.0.1.2: icmp_seq=8 ttl=63 time=0.109 ms
64 bytes from 10.0.1.2: icmp_seq=9 ttl=63 time=0.064 ms
64 bytes from 10.0.1.2: icmp_seq=10 ttl=63 time=0.210 ms
^C
--- 10.0.1.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9237ms
rtt min/avg/max/mdev = 0.062/0.132/0.399/0.097 ms
mininet>
```

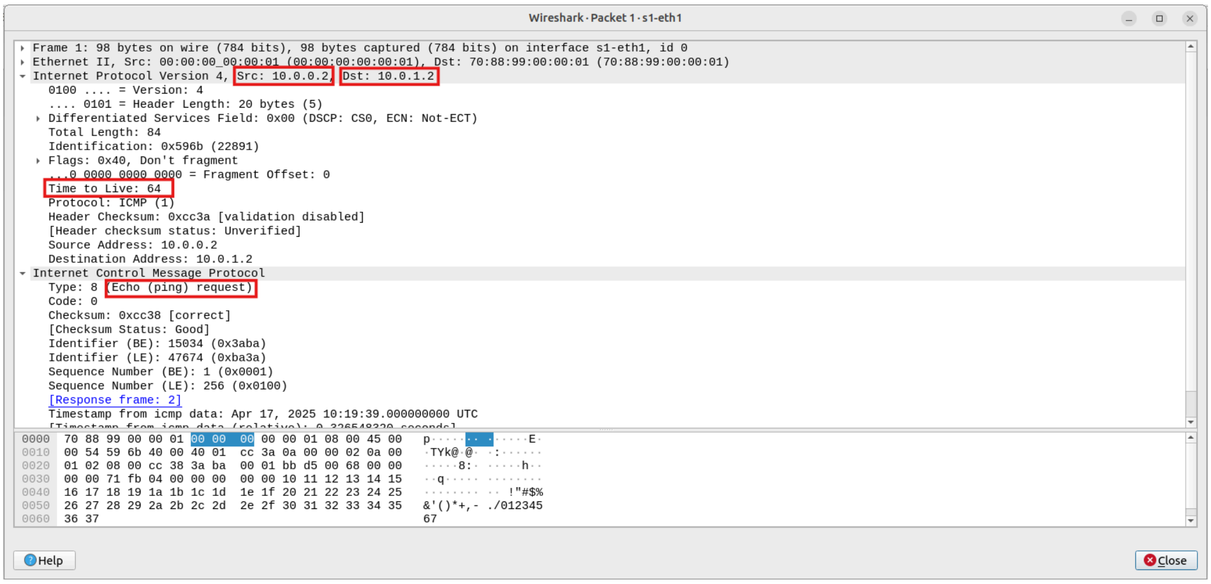
Y en Wireshark, vemos lo siguiente:



```
Apply a display filter … <Ctrl-/>
No.   Time          Source               Destination  Protocol  Length Info
    1 0.000000000   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=1/256, ttl=64 (reply in 2)
    2 0.000360776   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=1/256, ttl=63 (request in 1)
    3 1.028125403   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=2/512, ttl=64 (reply in 4)
    4 1.028181059   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=2/512, ttl=63 (request in 3)
    5 2.052499899   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=3/768, ttl=64 (reply in 6)
    6 2.052541535   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=3/768, ttl=63 (request in 5)
    7 3.084115662   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=4/1024, ttl=64 (reply in 8)
    8 3.084153628   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=4/1024, ttl=63 (request in 7)
    9 4.100183203   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=5/1280, ttl=64 (reply in 10)
   10 4.100215708   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=5/1280, ttl=63 (request in 9)
   11 5.124777265   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=6/1536, ttl=64 (reply in 12)
   12 5.124828868   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=6/1536, ttl=63 (request in 11)
   13 6.148415908   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=7/1792, ttl=64 (reply in 14)
   14 6.148466597   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=7/1792, ttl=63 (request in 13)
   15 7.172620177   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=8/2048, ttl=64 (reply in 16)
   16 7.172681414   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=8/2048, ttl=63 (request in 15)
   17 8.197030816   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=9/2304, ttl=64 (reply in 18)
   18 8.197062411   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=9/2304, ttl=63 (request in 17)
   19 9.241063514   10.0.0.2             10.0.1.2     ICMP        98 Echo (ping) request  id=0x3aba, seq=10/2560, ttl=64 (reply in 20)
   20 9.241176035   10.0.1.2             10.0.0.2     ICMP        98 Echo (ping) reply    id=0x3aba, seq=10/2560, ttl=63 (request in 19)
   21 44.241315958  fe80::4c8f:bfff:fed… ff02::fb     MDNS       203 Standard query 0x0000 PTR _nfs._tcp.local, "QM" question PTR _ipp._tcp.local, "QM" question PTR _ipps._tcp.local, "QM" question PTR …
   22 46.407342246  fe80::200:ff:fe00:1  ff02::2      ICMPv6      70 Router Solicitation from 00:00:00:00:00:01
   23 52.548849425  fe80::4c8f:bfff:fed… ff02::2      ICMPv6      70 Router Solicitation from 4e:8f:bf:df:69:30
```

Nos metemos en los dos primeros mensajes para inspeccionarlos.

El primero:

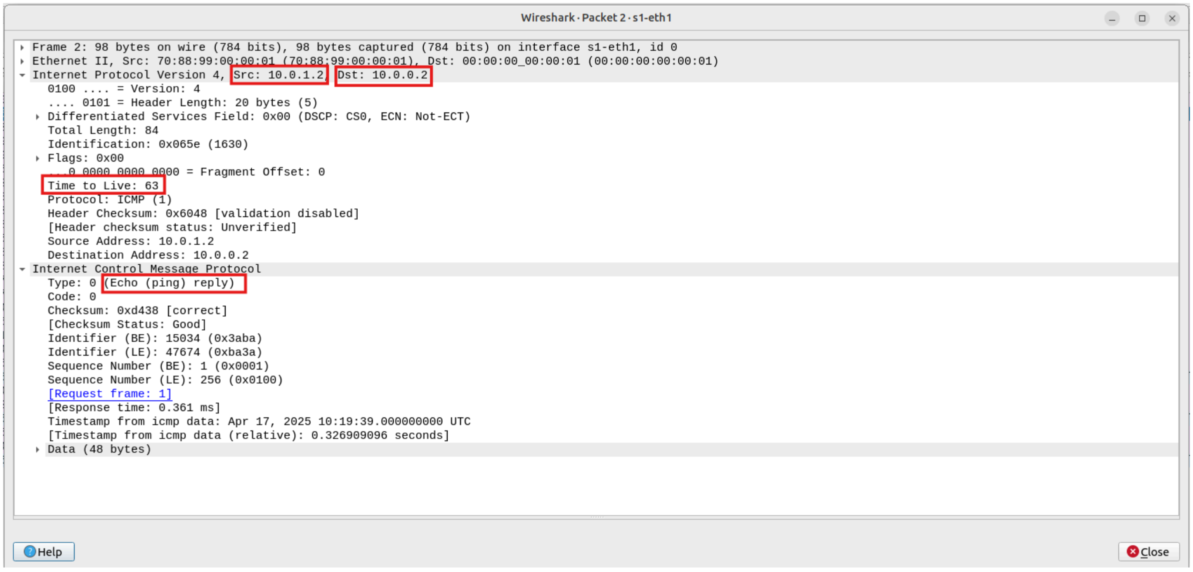


Wireshark · Packet 1 · s1-eth1

```
▸ Frame 1: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface s1-eth1, id 0
▸ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: 70:88:99:00:00:01 (70:88:99:00:00:01)
▾ Internet Protocol Version 4, Src: 10.0.0.2, Dst: 10.0.1.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x596b (22891)
  ▸ Flags: 0x40, Don't fragment
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: ICMP (1)
    Header Checksum: 0xcc3a [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.0.2
    Destination Address: 10.0.1.2
▾ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xcc38 [correct]
    [Checksum Status: Good]
    Identifier (BE): 15034 (0x3aba)
    Identifier (LE): 47674 (0xba3a)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Response frame: 2]
    Timestamp from icmp data: Apr 17, 2025 10:19:39.000000000 UTC
    [Timestamp from icmp data (relative): 0.326548320 seconds]

0000  70 88 99 00 00 01 00 00  00 00 00 01 08 00 45 00   p·····  ·· ·····E·
0010  00 54 59 6b 40 00 40 01  cc 3a 0a 00 00 02 0a 00   ·TYk@·@·  ·:······
0020  01 02 08 00 cc 38 3a ba  00 01 bb d5 00 68 00 00   ·····8:·  ·····h··
0030  00 00 71 fb 04 00 00 00  00 00 10 11 12 13 14 15   ··q·····  ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········  ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,-  ./012345
0060  36 37                                              67
```

Vemos que es de tipo "*echo request*" y que lleva especificadas las direcciones IPv4 con origen en 10.0.0.2 y con destino 10.0.1.2. Además, podemos ver que el TTL es de 64.

Lo que esperamos ver en el siguiente paquete será un *echo reply* con dirección IPv4 de origen 10.0.1.2 y de destino 10.0.0.2. Además, el TTL debería ser 63.



Wireshark · Packet 2 · s1-eth1

```
▸ Frame 2: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface s1-eth1, id 0
▸ Ethernet II, Src: 70:88:99:00:00:01 (70:88:99:00:00:01), Dst: 00:00:00_00:00:01 (00:00:00:00:00:01)
▾ Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.0.0.2
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 84
    Identification: 0x065e (1630)
  ▸ Flags: 0x00
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 63
    Protocol: ICMP (1)
    Header Checksum: 0x6048 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 10.0.1.2
    Destination Address: 10.0.0.2
▾ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xd438 [correct]
    [Checksum Status: Good]
    Identifier (BE): 15034 (0x3aba)
    Identifier (LE): 47674 (0xba3a)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Request frame: 1]
    [Response time: 0.361 ms]
    Timestamp from icmp data: Apr 17, 2025 10:19:39.000000000 UTC
    [Timestamp from icmp data (relative): 0.326909096 seconds]
  ▸ Data (48 bytes)
```

Efectivamente, lo que esperábamos ver es lo que hemos encontrado.

- **CÓDIGO vlanhost.py:**

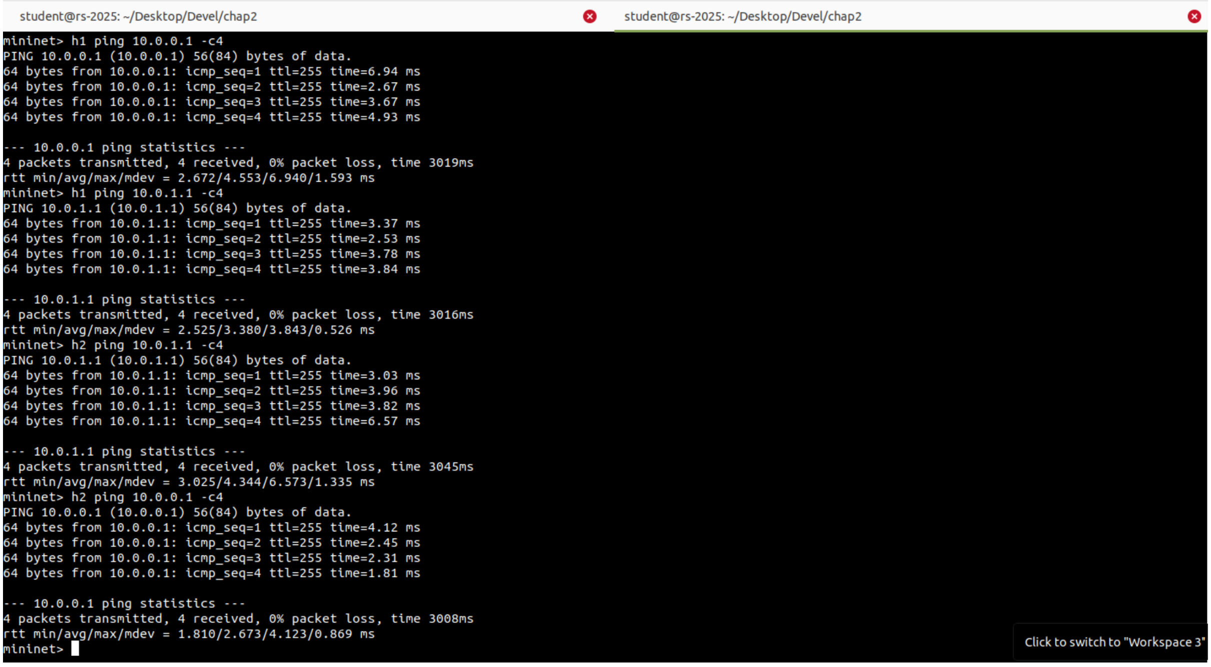
```
#!/usr/bin/env python3
"""
vlanhost.py: Host subclass that uses a VLAN tag for the default interface.

Dependencies:
    This class depends on the "vlan" package
    $ sudo apt-get install vlan

Usage (example uses VLAN ID=1000):
    From the command line:
        sudo mn --custom vlanhost.py --host vlan,vlan=1000

    From a script (see exampleUsage function below):
        from functools import partial
        from vlanhost import VLANHost

        ....

        host = partial( VLANHost, vlan=1000 )
        net = Mininet( host=host, ... )

    Directly running this script:
        sudo python vlanhost.py 1000

"""

import sys
from mininet.node import Host
from mininet.topo import Topo
from mininet.util import quietRun
from mininet.log import error
from mininet.node import RemoteController, OVSSwitch
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import SingleSwitchTopo
from mininet.log import setLogLevel

from functools import partial

class ARPHost( Host ):
    "Host connected to ARP interface"
    # Alternatively, one could define
    # def config(self, vlan=None, **params)
    # and avoid the
    # vlan = params.pop('vlan' None)
    # pylint: disable=arguments-differ
    def config( self, **params ):
        """Configure ARPHost according to (optional) parameters:
           arp: ARP ID for default interface"""

        arp = params.pop('arp', None)
        assert arp is not None, 'ARPHost without arp in instantiation'
        mac, ip=arp
        r = super().config(**params)
        self.setARP(ip, mac)

        return r

class LineTopo(Topo):
    def build(self):
        switch = self.addSwitch('s1')
        h1=self.addHost('h1', cls=ARPHost,
arp=('70:88:99:00:00:01','10.0.0.1'),ip='10.0.0.2/24',
mac="00:00:00:00:00:01", defaultRoute=('via 10.0.0.1'))
        h2=self.addHost('h2', cls=ARPHost,
arp=('70:88:99:10:00:02','10.0.1.1'),ip='10.0.1.2/24',
mac="00:00:00:00:00:02", defaultRoute=('via 10.0.1.1'))
        self.addLink(h1, switch)
        self.addLink(h2, switch)

def exampleCustomTags():
    setLogLevel( 'info' )
    """Simple example that exercises LineTopo"""
    net = Mininet( topo=LineTopo(), waitConnected=True,
switch=partial(OVSSwitch, protocols="OpenFlow13"),
controller=partial(RemoteController, ip="127.0.0.1"))
    net.start()
    CLI(net)
    net.stop()


if __name__ == '__main__':
    exampleCustomTags()
```

- **CÓDIGO L3Switch.py:**

```
# Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.ofproto import ether
from ryu.lib.packet import ipv4



class L3Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L3Switch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.router_mac = {
            '10.0.0.1': '70:88:99:00:00:01',
            '10.0.1.1': '70:88:99:10:00:02',
        }

    def forward_actions(self, parser, ofproto, port, src_mac, dst_mac):
        """Acciones para reenviar un paquete con ajustes de cabecera L3."""
        return [
            parser.OFPActionDecNwTtl(),
            parser.OFPActionSetField(eth_src=src_mac),
            parser.OFPActionSetField(eth_dst=dst_mac),
            parser.OFPActionOutput(port),
        ]

    def drop_actions(self, parser, ofproto):
        """Acciones para descartar un paquete."""
        return []

    def send_to_controller_actions(self, parser, ofproto):
        """Acciones para enviar un paquete al controlador."""
        return [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]




    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # Quitar LLDP
        match = parser.OFPMatch(eth_type=ether.ETH_TYPE_LLDP)
        self.add_flow(datapath, 10000, match, self.drop_actions(parser,
ofproto))

        #Quitar trafico IPv6
        match = parser.OFPMatch(eth_type=ether.ETH_TYPE_IPV6)
        self.add_flow(datapath, 10000, match, self.drop_actions(parser,
ofproto))

        #IPv4
        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.0.0', '255.255.255.0')
        )
        actions = self.forward_actions(
            parser, ofproto, 1,
            '70:88:99:00:00:01',  # MAC origen
            '00:00:00:00:00:01'   # MAC destino = h1
        )

        self.add_flow(datapath, 1000, match, actions)

        # IPv4
        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.1.0', '255.255.255.0')
        )
        actions = self.forward_actions(
            parser, ofproto, 2,
            '70:88:99:10:00:02',  # MAC origen
            '00:00:00:00:00:02'   # MAC destino
        )
        self.add_flow(datapath, 1000, match, actions)

        # 3. Regla por defecto: enviar al controlador (prioridad 0)
        match = parser.OFPMatch()
        self.add_flow(datapath, 0, match,
self.send_to_controller_actions(parser, ofproto))



    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # construct flow_mod message and send it.
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
match=match, instructions=inst)
        datapath.send_msg(mod)
```

Hito 2:

- Las modificaciones a simple_router.py
- Una captura de pantalla del terminal mininet con el comando `h2 ping -c4 10.0.1.1`



student@rs-2025: ~/Desktop/Devel/chap2
student@rs-2025: ~/Desktop/Devel/chap2
Click to switch to "Workspace 3"

```
mininet> h1 ping 10.0.0.1 -c4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=255 time=6.94 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=255 time=2.67 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=255 time=3.67 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=255 time=4.93 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3019ms
rtt min/avg/max/mdev = 2.672/4.553/6.940/1.593 ms
mininet> h1 ping 10.0.1.1 -c4
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=255 time=3.37 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=255 time=2.53 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=255 time=3.78 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=255 time=3.84 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 2.525/3.380/3.843/0.526 ms
mininet> h2 ping 10.0.1.1 -c4
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=255 time=3.03 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=255 time=3.96 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=255 time=3.82 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=255 time=6.57 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3045ms
rtt min/avg/max/mdev = 3.025/4.344/6.573/1.335 ms
mininet> h2 ping 10.0.0.1 -c4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=255 time=4.12 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=255 time=2.45 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=255 time=2.31 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=255 time=1.81 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3008ms
rtt min/avg/max/mdev = 1.810/2.673/4.123/0.869 ms
mininet> 
```

En esta captura de pantalla podemos ver los ping realizados desde el host 1 y host 2 hasta ambos puertos del encaminador. Efectivamente, comprobamos que los equipos finales pueden detectar ambos puertos del encaminador.

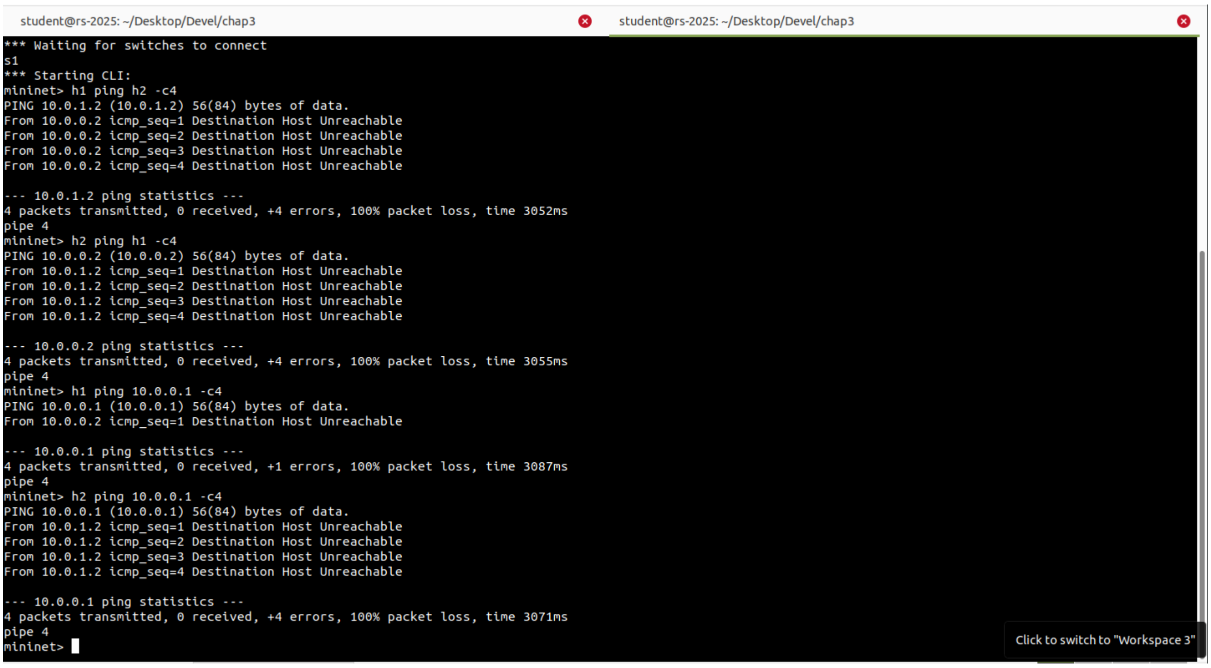
- **CÓDIGO L3Switch.py:**

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.ofproto import ether
from ryu.lib.packet import ipv4

from ryu.lib.packet import icmp
```

```
class L3Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L3Switch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.router_mac = {
            '10.0.0.1': '70:88:99:00:00:01',
            '10.0.1.1': '70:88:99:10:00:02',
        }

    def forward_actions(self, parser, ofproto, port, src_mac,
dst_mac):
        """Acciones para reenviar un paquete con ajustes de cabecera
L3."""
        return [
            parser.OFPActionDecNwTtl(),
            parser.OFPActionSetField(eth_src=src_mac),
            parser.OFPActionSetField(eth_dst=dst_mac),
            parser.OFPActionOutput(port),
        ]

    def drop_actions(self, parser, ofproto):
        """Acciones para descartar un paquete."""
        return []

    def send_to_controller_actions(self, parser, ofproto):
        """Acciones para enviar un paquete al controlador."""
        return [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]




    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser



        # Quitar LLDP
        match = parser.OFPMatch(eth_type=ether.ETH_TYPE_LLDP)
        self.add_flow(datapath, 10000, match,
self.drop_actions(parser, ofproto))

        #Quitar trafico IPv6
        match = parser.OFPMatch(eth_type=ether.ETH_TYPE_IPV6)
        self.add_flow(datapath, 10000, match,
self.drop_actions(parser, ofproto))


        match =parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.0.1', '255.255.255.255')
        )
        self.add_flow(datapath, 5000, match,
self.send_to_controller_actions(parser, ofproto))


        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.1.1', '255.255.255.255')
        )
        self.add_flow(datapath, 5000, match,
self.send_to_controller_actions(parser, ofproto))


        #IPv4
        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.0.0', '255.255.255.0')
        )
        actions = self.forward_actions(
            parser, ofproto, 1,
            '70:88:99:00:00:01',  # MAC origen
            '00:00:00:00:00:01'   # MAC destino = h1
        )

        self.add_flow(datapath, 1000, match, actions)

        # IPv4
        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.1.0', '255.255.255.0')
        )
        actions = self.forward_actions(
            parser, ofproto, 2,
            '70:88:99:10:00:02',  # MAC origen
            '00:00:00:00:00:02'   # MAC destino
        )
        self.add_flow(datapath, 1000, match, actions)

        # 3. Regla por defecto: enviar al controlador (prioridad 0)
        match = parser.OFPMatch()
        self.add_flow(datapath, 0, match,
self.send_to_controller_actions(parser, ofproto))

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        port = msg.match['in_port']
        pkt = packet.Packet(data=msg.data)
        self.logger.info("packet-in %s" % (pkt,))
        pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
        if not pkt_ethernet:
            return
        pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
        pkt_icmp = pkt.get_protocol(icmp.icmp)
        if pkt_icmp:
            self._handle_icmp(datapath, port, pkt_ethernet, pkt_ipv4,
pkt_icmp)
            return


    def _handle_icmp(self, datapath, port, pkt_ethernet, pkt_ipv4,
pkt_icmp):
        if pkt_icmp.type != icmp.ICMP_ECHO_REQUEST:
            return

        pkt = packet.Packet()

pkt.add_protocol(ethernet.ethernet(ethertype=pkt_ethernet.ethertype,
                                           dst=pkt_ethernet.src,

src=self.router_mac.get(pkt_ipv4.dst)))
        pkt.add_protocol(ipv4.ipv4(dst=pkt_ipv4.src,
                                   src=pkt_ipv4.dst,
                                   proto=pkt_ipv4.proto))
        pkt.add_protocol(icmp.icmp(type_=icmp.ICMP_ECHO_REPLY,
                                   code=icmp.ICMP_ECHO_REPLY_CODE,
                                   csum=0,
                                   data=pkt_icmp.data))
        #self.send_to_controller_actions(parser, ofproto)
        self._send_packet(datapath,port,pkt)

    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # construct flow_mod message and send it.
        inst =
[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
match=match, instructions=inst)
        datapath.send_msg(mod)

    def _send_packet(self, datapath, port, pkt):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        pkt.serialize()
        self.logger.info("packet-out %s" % (pkt,))
        data = pkt.data
        actions = [parser.OFPActionOutput(port=port)]
        out = parser.OFPPacketOut(datapath=datapath,
                                  buffer_id=ofproto.OFP_NO_BUFFER,
                                  in_port=ofproto.OFPP_CONTROLLER,
                                  actions=actions,
                                  data=data)

        datapath.send_msg(out)
```

Hito 3:

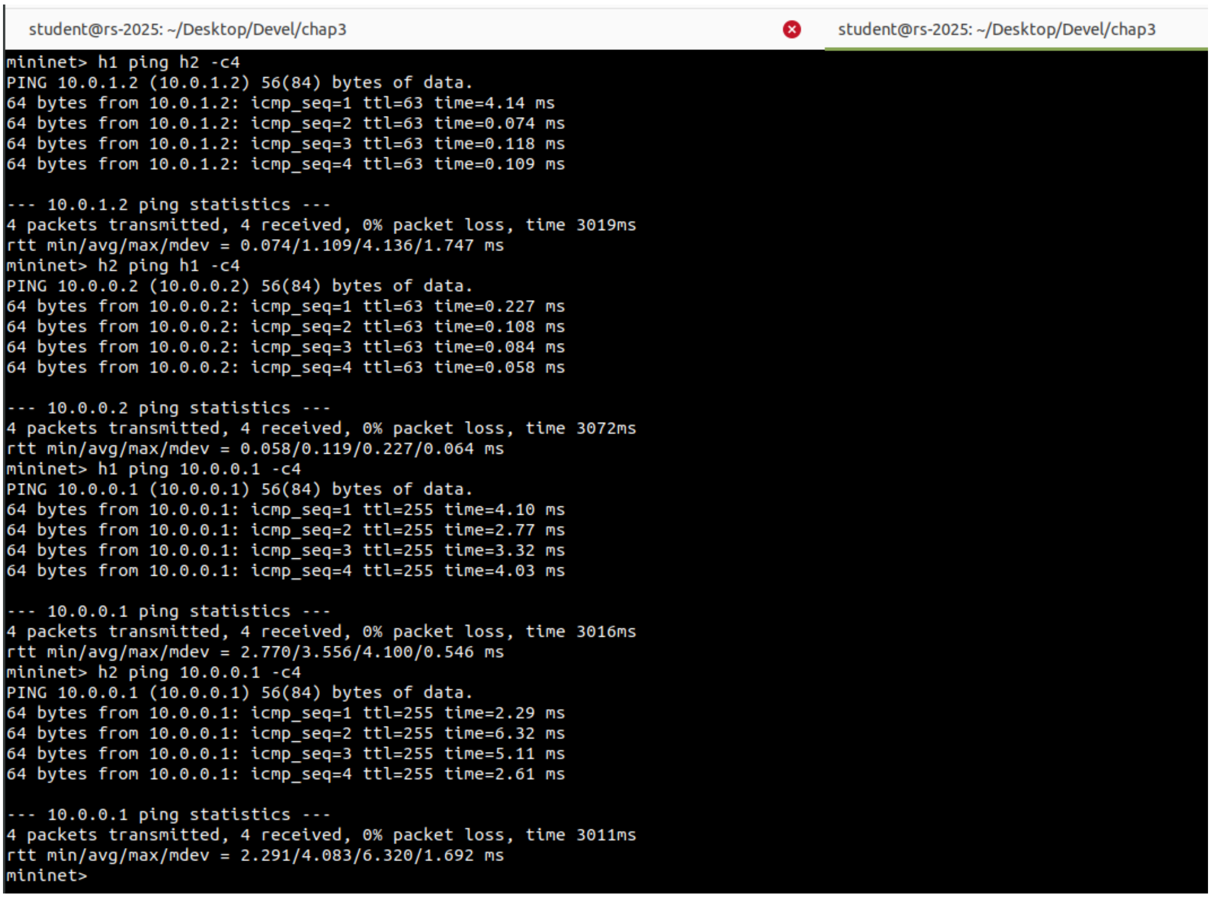
- Las modificaciones a scenario.py
- Las modificaciones a simple_router.py
- Una captura de pantalla de Wireshark con el tráfico para el comando `h1 ping -c 2 h2`
- Una captura de pantalla del terminal mininet para dicho comando


student@rs-2025: ~/Desktop/Devel/chap3
student@rs-2025: ~/Desktop/Devel/chap3
Click to switch to "Workspace 3"

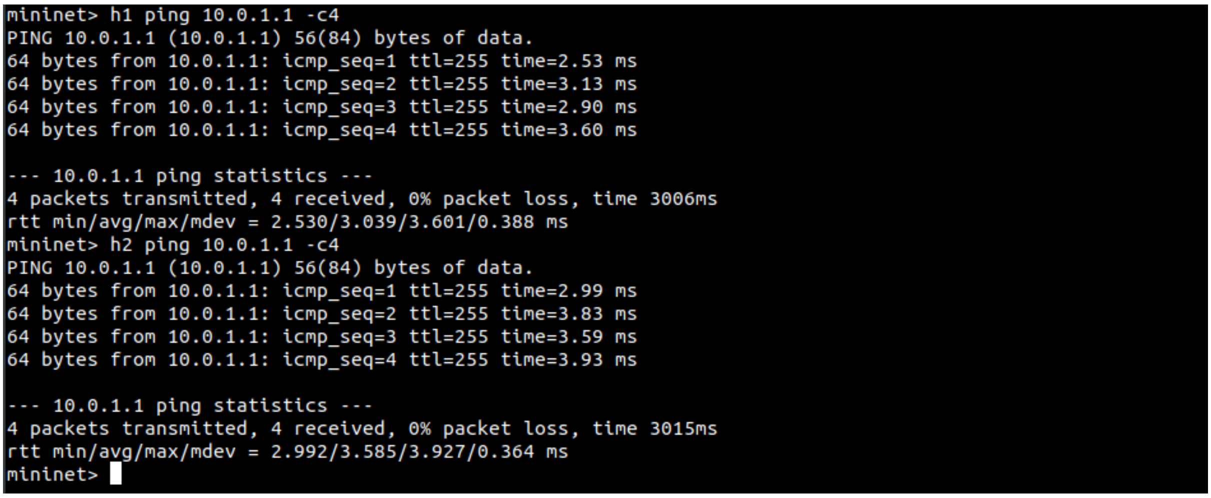
```
*** Waiting for switches to connect
s1
*** Starting CLI:
mininet> h1 ping h2 -c4
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable
From 10.0.0.2 icmp_seq=2 Destination Host Unreachable
From 10.0.0.2 icmp_seq=3 Destination Host Unreachable
From 10.0.0.2 icmp_seq=4 Destination Host Unreachable

--- 10.0.1.2 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3052ms
pipe 4
mininet> h2 ping h1 -c4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.1.2 icmp_seq=1 Destination Host Unreachable
From 10.0.1.2 icmp_seq=2 Destination Host Unreachable
From 10.0.1.2 icmp_seq=3 Destination Host Unreachable
From 10.0.1.2 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3055ms
pipe 4
mininet> h1 ping 10.0.0.1 -c4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.2 icmp_seq=1 Destination Host Unreachable

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 0 received, +1 errors, 100% packet loss, time 3087ms
pipe 4
mininet> h2 ping 10.0.0.1 -c4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.1.2 icmp_seq=1 Destination Host Unreachable
From 10.0.1.2 icmp_seq=2 Destination Host Unreachable
From 10.0.1.2 icmp_seq=3 Destination Host Unreachable
From 10.0.1.2 icmp_seq=4 Destination Host Unreachable

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3071ms
pipe 4
mininet> 
```

Al eliminar todo lo relacionado con la programación estática de la resolución de las direcciones MAC, es decir al eliminar todo lo relativo a ARP, y después, al ejecutar el controlador del capítulo anterior, vemos como no podemos hacer ping desde los equipos a cualquier otra dirección IP del escenario, tal y como se indica en la primera parte del hito.

Tras modificar el controlador (nuestro *L3Switch.py*)

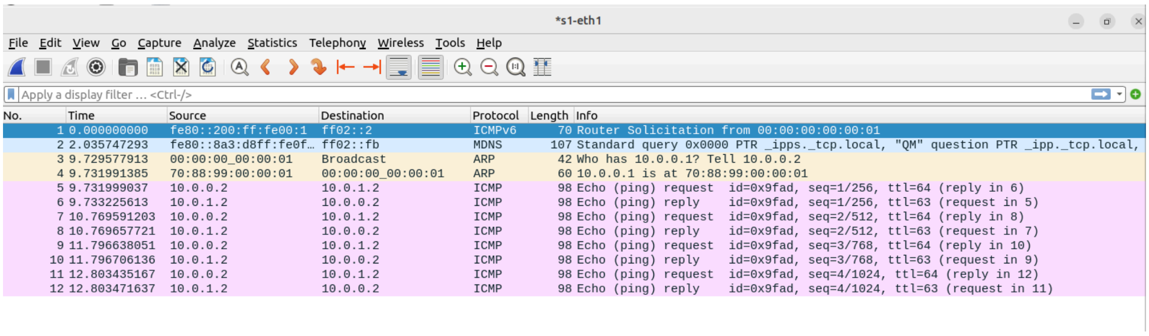

student@rs-2025: ~/Desktop/Devel/chap3
student@rs-2025: ~/Desktop/Devel/chap3

```
mininet> h1 ping h2 -c4
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=63 time=4.14 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=63 time=0.074 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=63 time=0.118 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=63 time=0.109 ms

--- 10.0.1.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3019ms
rtt min/avg/max/mdev = 0.074/1.109/4.136/1.747 ms
mininet> h2 ping h1 -c4
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=63 time=0.227 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=63 time=0.108 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=63 time=0.084 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=63 time=0.058 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3072ms
rtt min/avg/max/mdev = 0.058/0.119/0.227/0.064 ms
mininet> h1 ping 10.0.0.1 -c4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=255 time=4.10 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=255 time=2.77 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=255 time=3.32 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=255 time=4.03 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3016ms
rtt min/avg/max/mdev = 2.770/3.556/4.100/0.546 ms
mininet> h2 ping 10.0.0.1 -c4
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=255 time=2.29 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=255 time=6.32 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=255 time=5.11 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=255 time=2.61 ms

--- 10.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3011ms
rtt min/avg/max/mdev = 2.291/4.083/6.320/1.692 ms
mininet>
```



```
mininet> h1 ping 10.0.1.1 -c4
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=255 time=2.53 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=255 time=3.13 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=255 time=2.90 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=255 time=3.60 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 2.530/3.039/3.601/0.388 ms
mininet> h2 ping 10.0.1.1 -c4
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=255 time=2.99 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=255 time=3.83 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=255 time=3.59 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=255 time=3.93 ms

--- 10.0.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3015ms
rtt min/avg/max/mdev = 2.992/3.585/3.927/0.364 ms
mininet> 
```

Como podemos ver en las capturas, ahora sí es posible hacer ping a todos las direcciones.


*s1-eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::200:ff:fe00:1	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:00:00:01
2	2.035747293	fe80::8a3:d8ff:fe0f…	ff02::fb	MDNS	107	Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR _ipp._tcp.local,
3	9.729577913	00:00:00_00:00:01	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.2
4	9.731991385	70:88:99:00:00:01	00:00:00_00:00:01	ARP	60	10.0.0.1 is at 70:88:99:00:00:01
5	9.731999037	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request id=0x9fad, seq=1/256, ttl=64 (reply in 6)
6	9.733225613	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply id=0x9fad, seq=1/256, ttl=63 (request in 5)
7	10.769591203	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request id=0x9fad, seq=2/512, ttl=64 (reply in 8)
8	10.769657721	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply id=0x9fad, seq=2/512, ttl=63 (request in 7)
9	11.796638051	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request id=0x9fad, seq=3/768, ttl=64 (reply in 10)
10	11.796706136	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply id=0x9fad, seq=3/768, ttl=63 (request in 9)
11	12.803435167	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request id=0x9fad, seq=4/1024, ttl=64 (reply in 12)
12	12.803471637	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply id=0x9fad, seq=4/1024, ttl=63 (request in 11)

- **CÓDIGO vlanhost.py:**

```
#!/usr/bin/env python3

import sys
from mininet.node import Host
from mininet.topo import Topo
from mininet.util import quietRun
from mininet.log import error
from mininet.node import RemoteController, OVSSwitch
from mininet.net import Mininet
from mininet.cli import CLI
from mininet.topo import SingleSwitchTopo
from mininet.log import setLogLevel

from functools import partial



class LineTopo(Topo):
    def build(self):
        switch = self.addSwitch('s1')
        h1=self.addHost('h1', ip='10.0.0.2/24', mac="00:00:00:00:00:01",
defaultRoute=('via 10.0.0.1'))
        h2=self.addHost('h2', ip='10.0.1.2/24', mac="00:00:00:00:00:02",
defaultRoute=('via 10.0.1.1'))
        self.addLink(h1, switch)
        self.addLink(h2, switch)

def exampleCustomTags():
    setLogLevel( 'info' )
    net = Mininet( topo=LineTopo(), waitConnected=True,
switch=partial(OVSSwitch, protocols="OpenFlow13"),
controller=partial(RemoteController, ip="127.0.0.1"))
    net.start()
    CLI(net)
    net.stop()


if __name__ == '__main__':
    exampleCustomTags()
```

- **CÓDIGO L3Switch.py:**

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
#from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types
from ryu.ofproto import ether
#from ryu.lib.packet import ipv4
#from ryu.lib.packet import icmp
from ryu.lib.packet import ethernet, ipv4, icmp, arp


class L3Switch(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(L3Switch, self).__init__(*args, **kwargs)
        self.mac_to_port = {}
        self.router_mac = {
            '10.0.0.1': '70:88:99:00:00:01',
            '10.0.1.1': '70:88:99:10:00:02',
        }

    def forward_actions(self, parser, ofproto, port, src_mac,
dst_mac):
        """Acciones para reenviar un paquete con ajustes de cabecera
L3."""
        return [
            parser.OFPActionDecNwTtl(),
            parser.OFPActionSetField(eth_src=src_mac),
            parser.OFPActionSetField(eth_dst=dst_mac),
            parser.OFPActionOutput(port),
        ]

    def drop_actions(self, parser, ofproto):
        """Acciones para descartar un paquete."""
        return []

    def send_to_controller_actions(self, parser, ofproto):
        """Acciones para enviar un paquete al controlador."""
        return [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER)]
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):
        datapath = ev.msg.datapath
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser



        # Quitar LLDP
        match = parser.OFPMatch(eth_type=ether.ETH_TYPE_LLDP)
        self.add_flow(datapath, 10000, match,
self.drop_actions(parser, ofproto))

        #Quitar trafico IPv6
        match = parser.OFPMatch(eth_type=ether.ETH_TYPE_IPV6)
        self.add_flow(datapath, 10000, match,
self.drop_actions(parser, ofproto))


        match =parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.0.1', '255.255.255.255')
        )
        self.add_flow(datapath, 5000, match,
self.send_to_controller_actions(parser, ofproto))


        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.1.1', '255.255.255.255')
        )
        self.add_flow(datapath, 5000, match,
self.send_to_controller_actions(parser, ofproto))


        #IPv4
        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.0.0', '255.255.255.0')
        )
        actions = self.forward_actions(
            parser, ofproto, 1,
            '70:88:99:00:00:01',  # MAC origen
            '00:00:00:00:00:01'   # MAC destino = h1
        )

        self.add_flow(datapath, 1000, match, actions)

        # IPv4
        match = parser.OFPMatch(
            eth_type=ether.ETH_TYPE_IP,
            ipv4_dst=('10.0.1.0', '255.255.255.0')
        )
        actions = self.forward_actions(
            parser, ofproto, 2,
            '70:88:99:10:00:02',  # MAC origen
            '00:00:00:00:00:02'   # MAC destino
        )
        self.add_flow(datapath, 1000, match, actions)

        # 3. Regla por defecto: enviar al controlador (prioridad 0)
        match = parser.OFPMatch()
        self.add_flow(datapath, 0, match,
self.send_to_controller_actions(parser, ofproto))

    @set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
    def _packet_in_handler(self, ev):
        msg = ev.msg
        datapath = msg.datapath
        port = msg.match['in_port']
        pkt = packet.Packet(data=msg.data)
        self.logger.info("packet-in %s" % (pkt,))
        pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
        if not pkt_ethernet:
            return
        pkt_arp = pkt.get_protocol(arp.arp)
        if pkt_arp:
            self._handle_arp(datapath, port, pkt_ethernet, pkt_arp)
            return
        pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
        pkt_icmp = pkt.get_protocol(icmp.icmp)
        if pkt_icmp:
            self._handle_icmp(datapath, port, pkt_ethernet, pkt_ipv4,
pkt_icmp)
            return
    def _handle_icmp(self, datapath, port, pkt_ethernet, pkt_ipv4,
pkt_icmp):
        if pkt_icmp.type != icmp.ICMP_ECHO_REQUEST:
            return

        pkt = packet.Packet()

pkt.add_protocol(ethernet.ethernet(ethertype=pkt_ethernet.ethertype,
                                           dst=pkt_ethernet.src,

src=self.router_mac.get(pkt_ipv4.dst)))
        pkt.add_protocol(ipv4.ipv4(dst=pkt_ipv4.src,
                                   src=pkt_ipv4.dst,
                                   proto=pkt_ipv4.proto))
        pkt.add_protocol(icmp.icmp(type_=icmp.ICMP_ECHO_REPLY,
                                   code=icmp.ICMP_ECHO_REPLY_CODE,
                                   csum=0,
                                   data=pkt_icmp.data))
        #self.send_to_controller_actions(parser, ofproto)
        self._send_packet(datapath,port,pkt)

    def _handle_arp(self, datapath, port, pkt_ethernet, pkt_arp):
        if pkt_arp.opcode != arp.ARP_REQUEST:
            return
        pkt = packet.Packet()

pkt.add_protocol(ethernet.ethernet(ethertype=pkt_ethernet.ethertype,
                                           dst=pkt_ethernet.src,

src=self.router_mac.get(pkt_arp.dst_ip))) ######
        pkt.add_protocol(arp.arp(opcode=arp.ARP_REPLY,

src_mac=self.router_mac.get(pkt_arp.dst_ip),
                                 src_ip=pkt_arp.dst_ip,
                                 dst_mac=pkt_arp.src_mac,
                                 dst_ip=pkt_arp.src_ip))
        self._send_packet(datapath, port, pkt)



    def add_flow(self, datapath, priority, match, actions):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser

        # construct flow_mod message and send it.
        inst =
[parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
match=match, instructions=inst)
        datapath.send_msg(mod)

    def _send_packet(self, datapath, port, pkt):
        ofproto = datapath.ofproto
        parser = datapath.ofproto_parser
        pkt.serialize()
        self.logger.info("packet-out %s" % (pkt,))
        data = pkt.data
        actions = [parser.OFPActionOutput(port=port)]
        out = parser.OFPPacketOut(datapath=datapath,
                                  buffer_id=ofproto.OFP_NO_BUFFER,
                                  in_port=ofproto.OFPP_CONTROLLER,
                                  actions=actions,
                                  data=data)

        datapath.send_msg(out)
```