

# DESARROLLO DE UNA APLICACIÓN SDN

Jose María de la Cruz Sánchez – 100384116  
Jacobó del Castillo Monche - 100384048

## 1. Implementación básica de la conmutación de paquetes IPv4

En este laboratorio pretendemos emular el siguiente entorno:

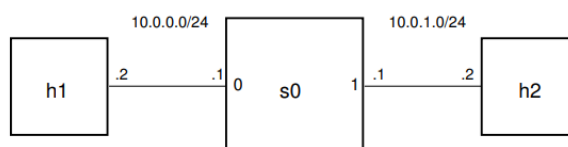


Figure 1.1.: Escenario Mininet del desarrollo

Adicionalmente, se supone la siguiente correspondencia entre direcciones IPv4 y MACs:

Dirección IPv4	MAC	Dirección IPv4	MAC
10.0.0.1	77:88:99:00:00:01	10.0.1.1	77:88:99:00:00:02
10.0.0.2	00:00:00:00:00:01	10.0.1.2	00:00:00:00:00:02

### HITO 1

Para capturar el tráfico en el extremo de h1 primero deberemos de ejecutar RYU-MANAGER con nuestro código 'simplerouter.py' y en otra terminal ejecutar nuestro 'scenario.py'.

Podemos verlo en las siguientes imágenes.

```
student@uc3m: ~/Desktop/lab3/hito1
File Edit View Search Terminal Help
student@uc3m:~/Desktop/lab3/hito1$ sudo ./scenario.py
*** Creating network
*** Adding controller
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
mininet>

student@uc3m: ~/Desktop/lab3/hito1
File Edit View Search Terminal Help
student@uc3m:~/Desktop/lab3/hito1$ ryu-manager simplerouter.py
loading app simplerouter.py
loading app ryu.controller.ofp_handler
instantiating app simplerouter.py of SimpleRouter
instantiating app ryu.controller.ofp_handler of OFPHandler
s1
```

### EJECUTAMOS H1 PING –C5 H2 Y VEMOS WHIRESHARK

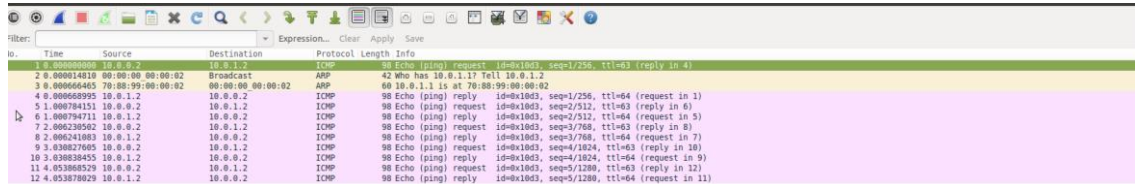
Para comprobar que el campo TTL del tráfico en el extremo de h1 decrementa debemos realizar un ping de h1 a h2.

```
mininet> h1 ping -c 5 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=0.49 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=63 time=0.843 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=63 time=0.838 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=63 time=0.863 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=63 time=0.835 ms

... 10.0.1.2 ping statistics ...
5 packets transmitted, 5 received, 0% packet loss, time 4815ms
rtt min/avg/max/mdev = 0.455/0.536/2.494/0.382 ms
mininet>
```

Por último, lanzamos WhireShark y capturamos el tráfico para comprobar el decremento del campo 'TTL'.

Podemos comprobarlo en la siguiente imagen:



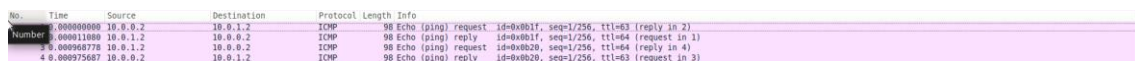
No.	Time	Source	Destination	Protocol	Length	Info
2	0.000014818	00:00:00:00:00:02	Broadcast	ARP	42	Who has 10.0.1.1? Tell 10.0.1.2
3	0.000066465	70:88:99:00:00:02	00:00:00:00:00:02	ARP	60	10.0.1.1 is at 70:88:99:00:00:02
4	0.000068995	10.0.1.2	10.0.1.2	ICMP	98	Echo (ping) reply 10.0.1.1, seq=1/256, ttl=63 (request in 1)
5	1.000794151	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request 10.0.1.1, seq=2/512, ttl=63 (reply in 6)
6	1.000794711	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply 10.0.1.1, seq=2/512, ttl=64 (request in 5)
7	2.000239582	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request 10.0.1.1, seq=3/768, ttl=63 (reply in 8)
8	2.000241083	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply 10.0.1.1, seq=3/768, ttl=64 (request in 7)
9	3.030827605	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request 10.0.1.1, seq=4/1024, ttl=63 (reply in 10)
10	3.030830455	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply 10.0.1.1, seq=4/1024, ttl=64 (request in 9)
11	4.053808529	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request 10.0.1.1, seq=5/1280, ttl=63 (reply in 12)
12	4.053878029	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply 10.0.1.1, seq=5/1280, ttl=64 (request in 11)

## 2. Gestión del protocolo ICMP

### HITO 2

Realizamos un 'pingall' para demostrar que todos los equipos finales pueden detectar ambos puertos del encaminador mediante el protocolo ICMP.

Podemos comprobarlo en la siguiente imagen:

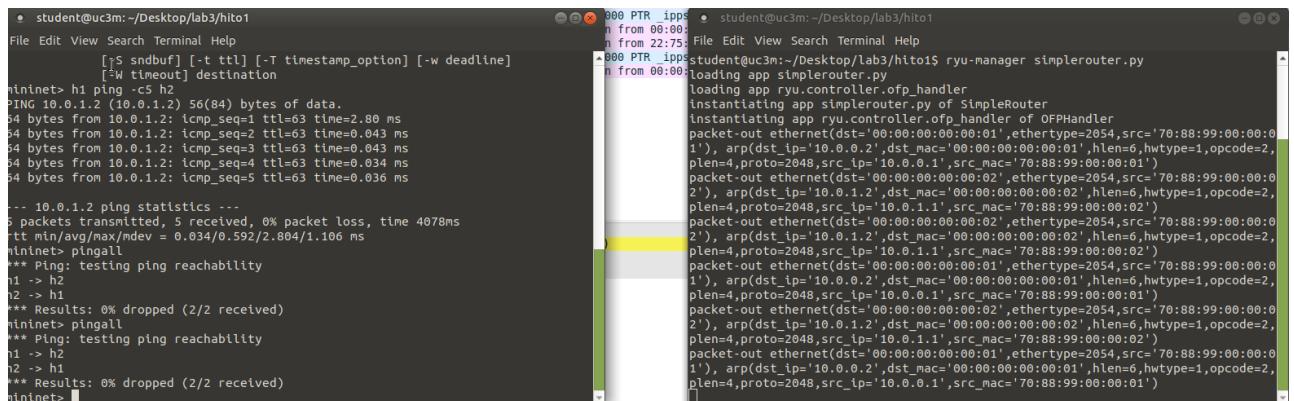


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) request 10.0.1.1, seq=1/256, ttl=63 (reply in 2)
2	0.000012080	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) reply 10.0.1.1, seq=1/256, ttl=64 (request in 1)
3	0.000968778	10.0.1.2	10.0.0.2	ICMP	98	Echo (ping) request 10.0.1.1, seq=2/512, ttl=63 (reply in 4)
4	0.000975687	10.0.0.2	10.0.1.2	ICMP	98	Echo (ping) reply 10.0.1.1, seq=2/512, ttl=64 (request in 3)

## 3. Integración del protocolo ARP

### HITO 3

En la siguiente imagen podemos observar que cuando arranca el escenario Mininet y el controlador, se puede realizar un ping de extremo a extremo y que las direcciones MAC se obtienen automáticamente sin necesidad de tenerlas programadas de manera estática en los equipos finales.



```
student@uc3m: ~/Desktop/lab3/hito1
File Edit View Search Terminal Help
[15 sndbuf] [-t ttl] [-T timestamp_option] [-w deadline]
[-W timeout] destination
mininet> h1 ping -c 5 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=2.80 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=63 time=0.043 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=63 time=0.043 ms
64 bytes from 10.0.1.2: icmp_seq=4 ttl=63 time=0.034 ms
64 bytes from 10.0.1.2: icmp_seq=5 ttl=63 time=0.036 ms
--- 10.0.1.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4078ms
rtt min/avg/max/ndev = 0.034/0.592/2.804/1.106 ms
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet>
```

## CODIGO USADO

### Scenario.py:

```
#!/usr/bin/env python

from mininet.node import Node
from mininet.node import RemoteController
from mininet.node import OVSSwitch
from functools import partial
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.log import setLogLevel
from mininet.cli import CLI

class SingleSwitchTopo(Topo):
    'We connect 1 single switch to n hosts'
    def build(self, N=1):
        switch = self.addSwitch('s1')
        for h in range(N):
            host=self.addHost('h%s'%(h+1),mac='00:00:00:00:00:%s'%(h+1), ip='10.0.%d,2/24'%h,defaultRoute='via 10.0.%d,1'%h)
            self.addLink(host, switch)

def simpleTestCLI():
    topo = SingleSwitchTopo(2)
    net = Mininet(topo, controller=partial(RemoteController, ip='127.0.0.1', port=6633), switch=partial(OVSSwitch, protocols="OpenFlow13"))
    net.start()

    CLI(net)
    net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    simpleTestCLI()
```

### Simplerouter.py:

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

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

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

    def __init__(self, *args, **kwargs):
        super(SimpleRouter, self).__init__(*args, **kwargs)

        self.hw_addr_p1 = '70:88:99:00:00:01'
        self.ip_addr_p1 = '10.0.0.1'

        self.hw_addr_p2 = '70:88:99:00:00:02'
        self.ip_addr_p2 = '10.0.1.1'

    def _controller_actions(self, parser, ofproto):
        return [
            parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                   ofproto.OFPCML_NO_BUFFER)
        ]

    def _forward_actions(self, parser, ofproto, port, src, dst):
```

```

def _forward_actions(self, parser, ofproto, port, src, dst):

    return [
        parser.OFPActionSetField(eth_src=src),
        parser.OFPActionSetField(eth_dst=dst),
        parser.OFPActionDecNwTtl(),
        parser.OFPActionOutput(port)
    ]

def _drop_actions(self, parser, ofproto):
    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=pkt_ethernet.dst))

    pkt.add_protocol(ipv4.ipv4(dst=pkt_ipv4.src,
                               src=pkt_ipv4.dst,
                               proto=pkt_ipv4.proto,
                               ttl=64))

    pkt.add_protocol(icmp.icmp(type=icmp.ICMP_ECHO_REPLY,
                               code=icmp.ICMP_ECHO_REPLY_CODE,
                               csum=0,
                               data=pkt_icmp.data))

    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()
    if pkt_ethernet.src == '00:00:00:00:00:01':
        pkt.add_protocol(ethernet.ethernet(ethertype=pkt_ethernet.ethertype,
                                         dst=pkt_ethernet.src,
                                         src=self.hw_addr_p1))
        pkt.add_protocol(arp.arp(opcode=arp.ARP_REPLY,
                                src_mac=self.hw_addr_p1,
                                src_ip=self.ip_addr_p1,
                                dst_mac=pkt_arp.src_mac,
                                dst_ip=pkt_arp.src_ip))
    else:
        pkt.add_protocol(ethernet.ethernet(ethertype=pkt_ethernet.ethertype,
                                         dst=pkt_ethernet.src,
                                         src=self.hw_addr_p2))
        pkt.add_protocol(arp.arp(opcode=arp.ARP_REPLY,
                                src_mac=self.hw_addr_p2,
                                src_ip=self.ip_addr_p2,
                                dst_mac=pkt_arp.src_mac,
                                dst_ip=pkt_arp.src_ip))

    self._send_packet(datapath, port, pkt)

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,

```



```

else:
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                             match=match, instructions=inst,
                             idle_timeout=idle_timeout)
    datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)

    #ICMP ECHO REQUEST
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(data=msg.data)
    pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
    pkt_arp = pkt.get_protocol(arp.arp)

    if not pkt_ethernet:
        return

    pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
    pkt_icmp = pkt.get_protocol(icmp.icmp)

    if pkt_arp:
        self.handle_arp(datapath, in_port, pkt_ethernet, pkt_arp)
        return

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)

    #ICMP ECHO REQUEST
    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(data=msg.data)
    pkt_ethernet = pkt.get_protocol(ethernet.ethernet)
    pkt_arp = pkt.get_protocol(arp.arp)

    if not pkt_ethernet:
        return

    pkt_ipv4 = pkt.get_protocol(ipv4.ipv4)
    pkt_icmp = pkt.get_protocol(icmp.icmp)

    if pkt_arp:
        self.handle_arp(datapath, in_port, pkt_ethernet, pkt_arp)
        return

    if pkt_icmp is not None:
        self.handle_icmp(datapath, in_port, pkt_ethernet, pkt_ipv4, pkt_icmp)
        return

```