

Universidade do Minho  
Departamento de Informática

Mestrado Integrado em Engenharia Informática

# Virtualização de Redes



---

## Trabalho Prático N<sup>o</sup>3 OpenFlow

---

A86617 Gonçalo Nogueira  
A82529 Carlos Afonso  
A74806 João Amorim

Braga  
Junho, 2021

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Respostas às questões</b>	<b>3</b>
2.1	Questão 1 . . . . .	3
2.2	Questão 2 . . . . .	4
2.3	Questão 3 . . . . .	5
2.4	Questão 4 . . . . .	6
2.5	Questão 5 . . . . .	7
<b>3</b>	<b>Exercício Prático 1</b>	<b>8</b>
3.1	Testes . . . . .	9
<b>4</b>	<b>Exercício Prático 2</b>	<b>11</b>
4.1	Topologia da rede . . . . .	11
4.2	Resolução . . . . .	12
4.2.1	Cenário 1 . . . . .	12
4.2.2	Cenário 2 . . . . .	13
4.2.3	Cenário 3 . . . . .	14
4.3	Testes . . . . .	15
<b>5</b>	<b>Conclusão</b>	<b>18</b>

# 1 Introdução

O presente relatório, realizado no âmbito da unidade curricular de Virtualização de Redes, destina-se inicialmente a apresentar um conjunto de perguntas e respostas relacionadas com o *OpenFlow*.

*OpenFlow* é um protocolo de comunicação que dá acesso ao plano de reencaminhamento de um switch de rede ou router pela rede de computadores.

O principal objetivo deste trabalho prático é a familiarização com este protocolo e com as funcionalidades da API POX.

Por fim, este relatório está dividido entre as respostas às questões formuladas no enunciado bem como a explicação das decisões tomadas pelo grupo para concretizar a parte prático do trabalho e os respectivos testes efetuados.

## 2 Respostas às questões

### 2.1 Questão 1

[ Q1: What type of OpenFlow message is used to retrieve this information? ]

Para obter as informações do estado lógico de um OpenFlow switch, são usadas as mensagens "Controller to switch", estas mensagens são iniciadas pelo controlador permitindo ao mesmo gerir o estado lógico de um switch incluindo a sua configuração e detalhes de entradas de tabelas de grupo e de fluxo.

Nesta categoria de mensagens inclui-se as **Read-State** usadas pelo controlador para coletar estatísticas das tabelas de fluxo do switch, portas e entradas individuais das tabelas e **Features** quando se estabelece a ligação entre controlador e switch, onde o switch responde com as suas capacidades suportadas.

```
goncalo@goncalo-VirtualBox:~/Desktop$ sudo ovs-ofctl show s1
[sudo] password for goncalo:
OpenFlow 1.3 switch 's1' (dpid:0000000000000001)
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(s1-eth1): addr:0e:26:3c:e3:75:e7
  config: 0
  state: 0
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:fe:78:14:3f:57:df
  config: 0
  state: 0
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
3(s1-eth3): addr:7a:b3:5e:cb:41:47
  config: 0
  state: 0
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
LOCAL(s1): addr:a2:df:0d:99:b1:4b
  config: PORT_DOWN
  state: LINK_DOWN
  speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
goncalo@goncalo-VirtualBox:~/Desktop$
```

## 2.2 Questão 2

Q2: Write the `ovs-ofctl` command to delete just the first Flow Table entry. Write the `ovs-ofctl` command to delete all entries of `s1`. (man `ovs-ofctl`)

Para eliminar apenas uma entrada o comando usado é **`sudo ovs-ofctl del-flows s1 in_port=1`** e para apagar todas as entradas basta apenas especificar o switch, sendo o comando **`sudo ovs-ofctl del-flows s1`**.

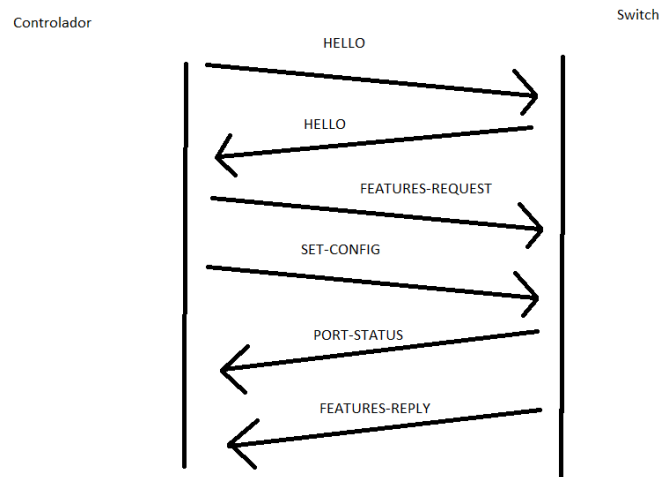
```
goncalo@goncalo-VirtualBox:~/Desktop$ sudo ovs-ofctl del-flows s1 in_port=1
goncalo@goncalo-VirtualBox:~/Desktop$ sudo ovs-ofctl dump-flows s1
 cookie=0x0, duration=19.832s, table=0, n_packets=0, n_bytes=0, in_port="s1-eth2" actions=output:"s1-eth1"
goncalo@goncalo-VirtualBox:~/Desktop$ sudo ovs-ofctl del-flows s1
goncalo@goncalo-VirtualBox:~/Desktop$ sudo ovs-ofctl dump-flows s1
goncalo@goncalo-VirtualBox:~/Desktop$
```

## 2.3 Questão 3

Q3: Describe the start up communication between switch and controller, draw a sequence diagram that shows the type of OpenFlow messages being exchanged. Explain their purpose. Present a print of the Wireshark capture.

Quando uma conexão *OpenFlow* é inicialmente estabelecida entre controlador e switch, cada lado da conexão deve imediatamente enviar uma mensagem *HELLO* onde o campo da versão está definido com a versão mais recente suportada pelo lado que envia a mensagem. Após a recepção desta mensagem, o recetor calcula a versão a utilizar fazendo a comparação entre a sua suportada e a da que recebeu e escolhendo a menor. Se a versão negociada for aceite a conexão procede, caso contrário envia resposta *OFPT\_ERROR* com o campo do tipo *OFPET\_HELLO\_FAILED*. Neste caso, e como se pode verificar no esquema, ambos os lados enviam uma mensagem *HELLO* para o outro lado, depois disso, o controlador envia um *OFPT\_FEATURES\_REQUEST* para conhecer as capacidades do switch e uma mensagem *OFPT\_SET\_CONFIG* para configurar o switch. O switch envia mensagem *OFPT\_PORT\_STATUS* para informar o controlador das suas portas e envia também a *OFPT\_FEATURES\_REPLY* como resposta ao pedido do controlador anteriormente.

55	19.131129432	127.0.0.1	127.0.0.1	TCP	66	57856	→ 6633 [ACK] Seq=1 Ack=1 Win=44032 Len=0 TSval=3753480446 TSecr=...
56	19.132365295	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO	
57	19.132375923	127.0.0.1	127.0.0.1	TCP	66	57856	→ 6633 [ACK] Seq=1 Ack=9 Win=44032 Len=0 TSval=3753480448 TSecr=...
58	19.132840417	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_HELLO	
59	19.132846311	127.0.0.1	127.0.0.1	TCP	66	6633	→ 57856 [ACK] Seq=9 Ack=9 Win=44032 Len=0 TSval=3753480448 TSecr=...
60	19.134420108	127.0.0.1	127.0.0.1	OpenFlow	74	Type: OFPT_FEATURES_REQUEST	
61	19.134426223	127.0.0.1	127.0.0.1	TCP	66	57856	→ 6633 [ACK] Seq=9 Ack=17 Win=44032 Len=0 TSval=3753480450 TSecr=...
62	19.134434040	127.0.0.1	127.0.0.1	OpenFlow	78	Type: OFPT_SET_CONFIG	
63	19.134436945	127.0.0.1	127.0.0.1	TCP	66	57856	→ 6633 [ACK] Seq=9 Ack=29 Win=44032 Len=0 TSval=3753480450 TSecr=...
64	19.135473989	127.0.0.1	127.0.0.1	OpenFlow	130	Type: OFPT_PORT_STATUS	
65	19.135479427	127.0.0.1	127.0.0.1	TCP	66	6633	→ 57856 [ACK] Seq=29 Ack=73 Win=44032 Len=0 TSval=3753480451 TSecr=...
66	19.135770061	127.0.0.1	127.0.0.1	OpenFlow	290	Type: OFPT_FEATURES_REPLY	
67	19.135774388	127.0.0.1	127.0.0.1	TCP	66	6633	→ 57856 [ACK] Seq=29 Ack=297 Win=44032 Len=0 TSval=3753480451 TSe...



## 2.4 Questão 4

Q4: Based on the captured packages explain, in your own words, what is happening.

Sendo a primeira vez que se está a tentar enviar pacotes entre dois hosts, não existe ainda nenhuma entrada na tabela de fluxo no switch, o que causa o envio de uma mensagem *OFPT\_PACKET\_IN* do switch para o controlador com o pacote em questão, pois o switch não sabe que rota escolher para reencaminhar o pacote.

O controlador após receber esta mensagem determina qual a melhor rota para reencaminhar o pacote e envia essa informação para o switch através da mensagem *OFPT\_FLOW\_MOD* que adiciona essa informação na tabela de fluxos conhecendo agora a ação a tomar para reencaminhar pacotes com características semelhantes.

Por fim o controlador, envia uma mensagem *OFPT\_PACKET\_OUT* novamente com o pacote que lhe foi enviado anteriormente, pois agora o switch já é capaz de o reencaminhar.

90	110.254201330	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=409 Ack=501 Win=86 Len=0 TSval=3755506314 TSecr...
91	110.254706217	127.0.0.1	127.0.0.1	OpenFlow	74 Type: OFPT_ECHO_REQUEST
92	110.254971237	127.0.0.1	127.0.0.1	OpenFlow	74 Type: OFPT_ECHO_REPLY
93	110.254994094	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=417 Ack=509 Win=86 Len=0 TSval=3755506314 TSecr...
94	110.376923442	00:00:00_00:00:01	00:00:00_00:00:02	OpenFlow	126 Type: OFPT_PACKET_IN
95	110.376977872	00:00:00_00:00:02	00:00:00_00:00:01	OpenFlow	126 Type: OFPT_PACKET_IN
96	110.377239074	127.0.0.1	127.0.0.1	OpenFlow	146 Type: OFPT_FLOW_MOD
97	110.377262021	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=537 Ack=589 Win=86 Len=0 TSval=3755506436 TSecr...
98	110.377876036	00:00:00_00:00:01	00:00:00_00:00:02	OpenFlow	132 Type: OFPT_PACKET_OUT
99	110.377892972	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=537 Ack=655 Win=86 Len=0 TSval=3755506437 TSecr...
100	110.377938480	127.0.0.1	127.0.0.1	OpenFlow	146 Type: OFPT_FLOW_MOD
101	110.377948666	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=537 Ack=735 Win=86 Len=0 TSval=3755506437 TSecr...
102	110.377967856	00:00:00_00:00:02	00:00:00_00:00:01	OpenFlow	132 Type: OFPT_PACKET_OUT
103	110.377976300	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=537 Ack=801 Win=86 Len=0 TSval=3755506437 TSecr...
104	110.379874072	00:00:00_00:00:02	00:00:00_00:00:01	OpenFlow	126 Type: OFPT_PACKET_IN
105	110.379916731	00:00:00_00:00:01	00:00:00_00:00:02	OpenFlow	126 Type: OFPT_PACKET_IN
106	110.380055889	127.0.0.1	127.0.0.1	OpenFlow	146 Type: OFPT_FLOW_MOD
107	110.380078334	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=657 Ack=881 Win=86 Len=0 TSval=3755506439 TSecr...
108	110.380115746	00:00:00_00:00:02	00:00:00_00:00:01	OpenFlow	132 Type: OFPT_PACKET_OUT
109	110.380125705	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=657 Ack=947 Win=86 Len=0 TSval=3755506439 TSecr...
110	110.380160104	127.0.0.1	127.0.0.1	OpenFlow	146 Type: OFPT_FLOW_MOD
111	110.380168899	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=657 Ack=1027 Win=86 Len=0 TSval=3755506439 TSecr...
112	110.380187227	00:00:00_00:00:01	00:00:00_00:00:02	OpenFlow	132 Type: OFPT_PACKET_OUT
113	110.380195255	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=657 Ack=1093 Win=86 Len=0 TSval=3755506439 TSecr...
114	115.381673764	127.0.0.1	127.0.0.1	OpenFlow	74 Type: OFPT_ECHO_REQUEST
115	115.381963045	127.0.0.1	127.0.0.1	OpenFlow	74 Type: OFPT_ECHO_REPLY
116	115.382004300	127.0.0.1	127.0.0.1	TCP	66 57856 -- 6633 [ACK] Seq=665 Ack=1101 Win=86 Len=0 TSval=3755506441 TSecr...

## 2.5 Questão 5

**Q5:** If you keep doing pings *mininet> h1 ping -c1 h2* you will not see any more *OFPT\_PACKET\_IN* and *OFPT\_PACKET\_OUT* packages capture by Wireshark. Why is this happening? (Use the command `$ sudo ovs-ofctl dump-flows s1` to justify our answer).

Não existem mais *OFPT\_PACKET\_IN* e *OFPT\_PACKET\_OUT*, devido após a primeira troca de pacotes, já se adicionou a ação a tomar na tabela de fluxo do switch, sendo assim deixa de ser preciso existir mensagens entre controlador e switch, pois o switch já tem a informação da rota pelo qual tem de encaminhar os pacotes.

Pode-se observar as entradas na tabela de fluxo do switch da seguinte forma:

```
gongch@gongchali:~/mininet$ cd ~/Desktop/tp3/code$ sudo ovs-ofctl dump-flows s1
cookie=0x0, duration=47.933s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,arp,in_port="s1-eth2",vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=2 actions=output:"s1-eth1"
cookie=0x0, duration=42.715s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,arp,in_port="s1-eth2",vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,arp_spa=10.0.0.2,arp_tpa=10.0.0.1,arp_op=1 actions=output:"s1-eth1"
cookie=0x0, duration=42.713s, table=0, n_packets=0, n_bytes=0, idle_timeout=60, priority=65535,arp,in_port="s1-eth1",vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,arp_spa=10.0.0.1,arp_tpa=10.0.0.2,arp_op=2 actions=output:"s1-eth2"
cookie=0x0, duration=47.933s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, priority=65535,icmp,in_port="s1-eth1",vlan_tci=0x0000,d_l_src=00:00:00:00:00:01,d_l_dst=00:00:00:00:00:02,nw_src=10.0.0.1,nw_dst=10.0.0.2,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth2"
cookie=0x0, duration=47.933s, table=0, n_packets=5, n_bytes=490, idle_timeout=60, priority=65535,icmp,in_port="s1-eth2",vlan_tci=0x0000,d_l_src=00:00:00:00:00:02,d_l_dst=00:00:00:00:00:01,nw_src=10.0.0.2,nw_dst=10.0.0.1,nw_tos=0,icmp_type=0,icmp_code=0 actions=output:"s1-eth1"
```



### 3 Exercício Prático 1

O objetivo deste exercício é criar um controlador *OpenFlow* que funciona como um *L2 MAC-learning switch*. Inicialmente o controlador funciona como um *Hub* ou seja, quando recebe um pacote com determinado destino faz um *Flooding* para a rede e eventualmente o pacote chega ao destino, pois este pacote é enviado para todo o lado sendo ignorado pelos hosts caso o endereço de destino seja diferente do seu. Esta forma de reencaminhar pacotes é muito pouco eficiente e enquanto os frames estão a ser enviados para todos os hosts a rede não pode ser utilizada para outras trocas de informação ficando indisponível por algum tempo até estes frames serem recebidos e descartados. Esta forma apresenta também grandes problemas de segurança e podem também haver colisões que tornam os frames impossíveis de ler levando a um reset da rede e um novo envio de frames.

Com a introdução de um switch, quando um host envia um frame que chega ao switch, é verificado qual o *endereço MAC de destino*, posteriormente consulta-se a tabela que associa estes endereços a uma determinada porta pela qual o switch está conectado ao host com esse endereço. Após estas verificações o switch reencaminha o pacote pela porta correta para o host destino (CAM table). Esta forma é bastante mais eficiente e usa muito pouco a *bandwidth* disponível da rede devido à inexistência de *flooding* da rede.

Para atingir o objetivo proposto, foi preciso associar num dicionário a porta que corresponde a cada endereço MAC do pacote de Input caso o pacote seja do tipo *ARP*.

Posteriormente cria-se um objeto *ofp\_match* verificando se o endereço MAC a enviar já tem uma porta associada no enunciado, caso isso seja nulo o comportamento a adotar é igual ao do Hub e o pacote é enviado por flooding para a rede. Caso exista, é preciso criar um objeto *ofp\_flow\_mod* para introduzir a rota determinada na tabela de fluxo para posterior uso caso o mesmo host envie novamente pacotes com o mesmo host destino.

```
def process_packet (self, packet, packet_in):
    # We want to output to all ports -- we do that using the special
    # OFPP_ALL port as the output port. (We could have also used
    # OFPP_FLOOD.)

    src_mac_address = str(packet.src)
    dst_mac_address = str(packet.dst)
    input_port = packet_in.in_port

    if (src_mac_address not in self.mac_to_port) and (packet.type == packet.ARP_TYPE):
        self.mac_to_port[src_mac_address] = input_port

    if dst_mac_address in self.mac_to_port:
        out_port = self.mac_to_port[dst_mac_address]
        msg = of.ofp_flow_mod()
        msg.match = of.ofp_match()
        msg.match.in_port = input_port
        msg.match.dl_dst = packet.dst
        msg.buffer_id = packet_in.buffer_id
        action = of.ofp_action_output(port = out_port)
        msg.actions.append(action)
        self.connection.send(msg)
        self.resend_packet(packet_in, out_port)
    else :
        self.resend_packet(packet_in, of.OFPP_ALL)
```

### 3.1 Testes

Com o comportamento inicial caso fosse efetuado um ping do host 1 para o host 2 com endereço IP 10.0.0.2, tanto o host 2 como 3 recebem os mesmos pacotes ARP e ICMP como se pode observar.

```
"Node: h2"                                     "Node: h3"
root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code# tcpdump -XX -n -i h2-et
h0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:35:19.060966 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 62390, seq 1, leng
th 64
    0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  .....E.
    0x0010:  0054 3702 4000 4001 8fa4 0a00 0001 0a00  .T..@.
    0x0020:  0002 0800 184f f3b6 0001 e7db d060 0000  .....0.
    0x0030:  0000 74e9 0000 0000 0000 1011 1213 1415  ....t.
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....!"#%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637                                     67
19:35:19.061024 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 62390, seq 1, length
64
    0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  .....E.
    0x0010:  0054 33a5 0000 4001 3302 0a00 0002 0a00  .T3...@.3.
    0x0020:  0001 0000 204f f3b6 0001 e7db d060 0000  .....0.
    0x0030:  0000 74e9 0000 0000 0000 1011 1213 1415  ....t.
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....!"#%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637                                     67
19:35:24.134184 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0000 0a00 0001                                     .....
19:35:24.137649 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
    0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0002 0a00 0002                                     .....
19:35:24.138237 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0002 0a00 0002                                     .....
19:35:24.138254 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0001 0a00 0001                                     .....
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code#

root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code# tcpdump -XX -n -i h3-eth0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:35:19.060967 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 62390, seq 1, length 64
    0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  .....E.
    0x0010:  0054 3702 4000 4001 8fa4 0a00 0001 0a00  .T..@.
    0x0020:  0002 0800 184f f3b6 0001 e7db d060 0000  .....0.
    0x0030:  0000 74e9 0000 0000 0000 1011 1213 1415  ....t.
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....!"#%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637                                     67
19:35:19.061789 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 62390, seq 1, length 64
    0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  .....E.
    0x0010:  0054 33a5 0000 4001 3302 0a00 0002 0a00  .T3...@.3.
    0x0020:  0001 0000 204f f3b6 0001 e7db d060 0000  .....0.
    0x0030:  0000 74e9 0000 0000 0000 1011 1213 1415  ....t.
    0x0040:  1617 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....!"#%
    0x0050:  2627 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637                                     67
19:35:24.138069 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0000 0a00 0001                                     .....
19:35:24.137670 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
    0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0002 0a00 0002                                     .....
19:35:24.138257 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0000 0a00 0002                                     .....
19:35:24.139509 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0001 0a00 0001                                     .....
^C
6 packets captured
6 packets received by filter
0 packets dropped by kernel
root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code#
```

Com as alterações feitas apenas o host destino recebe os pacotes.

```

"Node: h2"
18 packets captured
18 packets received by filter
0 packets dropped by kernel
root@goncalo-VirtualBox:~/home/goncalo/Desktop/tp3/code# tcpdump -XX -n -i h2-et
h0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h2-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:31:54.705636 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 62291, seq 1, leng
th 64
    0x0000:  0000 0000 0002 0000 0000 0001 0800 4500  .....E.
    0x0010:  0054 f2b9 4000 4001 33ad 0a00 0001 0a00  .T..@.3.....
    0x0020:  0002 0800 b7d8 f353 0001 1adb d060 0000  .....S.....
    0x0030:  0000 98c3 0a00 0000 0000 1011 1213 1415  .....
    0x0040:  1517 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....!""#$%
    0x0050:  2527 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637                                     67
19:31:54.705668 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 62291, seq 1, length
64
    0x0000:  0000 0000 0001 0000 0000 0002 0800 4500  .....E.
    0x0010:  0054 05bf 0000 4001 60e8 0a00 0002 0a00  .T....@.....
    0x0020:  0001 0000 b7d8 f353 0001 1adb d060 0000  .....S.....
    0x0030:  0000 98c3 0a00 0000 0000 1011 1213 1415  .....
    0x0040:  1517 1819 1a1b 1c1d 1e1f 2021 2223 2425  .....!""#$%
    0x0050:  2527 2829 2a2b 2c2d 2e2f 3031 3233 3435  &'()*+,-./012345
    0x0060:  3637                                     67
19:31:59.847675 ARP, Request who-has 10.0.0.1 tell 10.0.0.2, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0000 0a00 0001  .....
19:31:59.848056 ARP, Request who-has 10.0.0.2 tell 10.0.0.1, length 28
    0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0001 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0000 0a00 0002  .....
19:31:59.848056 ARP, Reply 10.0.0.1 is-at 00:00:00:00:00:01, length 28
    0x0000:  0000 0000 0002 0000 0000 0001 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0001 0a00 0001  .....
    0x0020:  0000 0000 0002 0a00 0002  .....
19:31:59.848056 ARP, Reply 10.0.0.2 is-at 00:00:00:00:00:02, length 28
    0x0000:  0000 0000 0001 0000 0000 0002 0806 0001  .....
    0x0010:  0800 0604 0002 0000 0000 0002 0a00 0002  .....
    0x0020:  0000 0000 0001 0a00 0001  .....
^C
6 packets captured

"Node: h3"
root@goncalo-VirtualBox:~/home/goncalo/Desktop/tp3/code# tcpdump -XX -n -i h3-et
h0
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on h3-eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
19:32:53.094921 IP6 fe80::200:ff:fe00:1 > ff02::2: ICMP6, router solicitation, l
ength 16
    0x0000:  3333 0000 0002 0000 0000 0001 86dd 6000  33.....
    0x0010:  0000 0010 3aff fe80 0000 0000 0000 0200  .....
    0x0020:  00ff fe00 0001 ff02 0000 0000 0000 0000  .....
    0x0030:  0000 0000 0002 8500 7b2c 0000 0000 0101  .....{.....
    0x0040:  0000 0000 0001  .....
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
root@goncalo-VirtualBox:~/home/goncalo/Desktop/tp3/code#

```

## 4 Exercício Prático 2

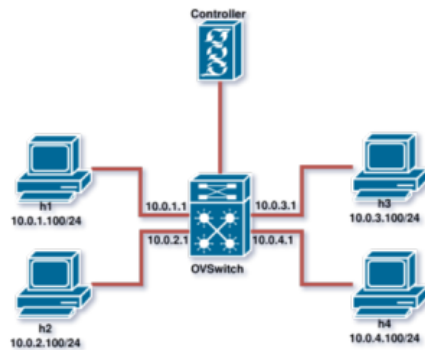
Neste exercício o objetivo é construir um *forwarder/switch* estático com 3 layers.

Geralmente um *router* tem de responder a *ARP Requests* e inicialmente haverá broadcasts que serão reencaminhados para o controlador. O objetivo é construir *ARP Replies* e enviar pelas portas corretas.

### 4.1 Topologia da rede

A construção da topologia de rede foi feita da seguinte forma:

```
class VrEx2Topo( Topo ):  
    def build( self ):  
        # Add hosts and switches  
        switch = self.addSwitch( 's1' )  
        host1 = self.addHost( 'h1', ip="10.0.1.100/24", defaultRoute = "via 10.0.1.1" )  
        host2 = self.addHost( 'h2', ip="10.0.2.100/24", defaultRoute = "via 10.0.2.1" )  
        host3 = self.addHost( 'h3', ip="10.0.3.100/24", defaultRoute = "via 10.0.3.1" )  
        host4 = self.addHost( 'h4', ip="10.0.4.100/24", defaultRoute = "via 10.0.4.1" )  
        # Add links  
        self.addLink( switch, host1 )  
        self.addLink( switch, host2 )  
        self.addLink( switch, host3 )  
        self.addLink( switch, host4 )  
  
topos = { 'vrex2topo': ( lambda: VrEx2Topo() ) }
```



## 4.2 Resolução

Inicialmente o controlador não tem capacidade para lidar com pacotes IP, sendo necessário completar o código dado para lidar com três cenários diferentes após um pacote IP recebido:

- O endereço IP de destino é um dos interfaces do router.
- O router não conhece o endereço MAC do endereço IP de destino.
- O router já conhece o endereço MAC do endereço IP de destino.

Antes de especificar os cenários individualmente descobre-se os endereços IP e MAC da interface do router com a *destination\_network* e com auxílio da tabela de routing e tabela de arp.

Encontra-se também a respetiva porta de output que corresponde á rede de destino.

```
##### EXERCISE 2 STARTS HERE #####
ip_addr_inter = self.routing_table[str(destination_network)][ 'RouterInterface' ]
mac_addr_inter = EthAddr(self.arp_table[ip_addr_inter])
output_port = self.routing_table[str(destination_network)][ 'Port' ]
if routable:
```

### 4.2.1 Cenário 1

Caso se verifique este cenário é apenas verificado se o protocolo do pacote recebido é do tipo ipv4 e caso isso se verifique é chamada o método **ICMP\_Handler**.

```
output_port = self.routing_table[str(destination_network)][ 'Port' ]
if routable:
    log.debug('PACKET IS ROUTABLE!')
    if ip_addr_inter == destination_ip :
        if ip_packet.protocol == ipv4.ICMP_PROTOCOL :
            self.ICMP_Handler(etherFrame,packet_in)
            log.debug('FIRST SCENARIO')
```

#### 4.2.2 Cenário 2

Este cenário é o pior caso que pode acontecer em termos de processamento pois é o que é preciso criar pacotes ARP e Ethernet Frames que serão depois enviados.

Inicialmente guarda-se no buffer o pacote e a rede de destino até a ARP Reply chegar.

Posteriormente é criado um pacote ARP do tipo Request onde a fonte é o endereço IP da interface de saída do router e o destino é o mesmo endereço IP do pacote IP inicialmente recebido. É necessário estabelecer também os endereços MAC deste pacote, sendo o da fonte o endereço MAC da interface de saída do router e o destino como ainda não é conhecido fico com o endereço MAC só com zeros.

Após a criação do pacote ARP é preciso criar um ethernet frame que irá transportar o pacote ARP criado. O endereço MAC fonte é o mesmo do que o ARP e o de destino é o endereço reservado para o broadcast.

Por fim, adiciona-se o pacote ARP ao payload do frame e envia-se usando a porta de saída que corresponde á rede destino.

```
else :
    # ARP if host MAC Address is not present -- the router does not know the mac addr of the ip dst
    if destination_ip not in self.arp_table:
        # Push frame to buffer
        # this packet is stored until the ARP reply arrives
        self.buffer[destination_ip] = {'IP_Packet': ip_packet, 'DestinationNetwork': destination_network}

        # Construct ARP Packet
        arpl = arp()
        arpl.opcode = arp.REQUEST # it is an ARP request after all
        arpl.protosrc = IPAddr(ip_addr_inter) # to the ip address of the router outgoing interface, you can use
        arpl.protodst = IPAddr(destination_ip) # var of your arp object to the destination ip address
        # NOTE: you can use the method IPAddr(destination_ip) to parse a string to an IP addr.

        arpl.hwsrc = mac_addr_inter # var of your arp object to the mac address of the router outgoing interface
        arpl.hwdst = EthAddr('00:00:00:00:00:00') # var of your arp object to an all zeros mac address '00:00:00:00:00:00'
        # NOTE: you can use the method EthAddr('00:00:00:00:00:00') to parse a string to an MAC addr.

        # Construct the ethernet frame
        ether1 = ethernet()
        ether1.type = ether1.ARP_TYPE # defines that this frame carries a ARP type message

        # now we need to define the mac address for the source of this communication, the mac address of the outgoing interface
        ether1.src = mac_addr_inter
        # the destination address is a broadcast mac address: 'FF:FF:FF:FF:FF:FF'
        ether1.dst = EthAddr('FF:FF:FF:FF:FF:FF')
        # NOTE: you can use the method EthAddr('FF:FF:FF:FF:FF:FF') to parse a string to an MAC addr.

        # Last, we attach the <your_arp_request_object> as the payload of the ethernet frame
        ether1.payload = arpl
        # Then we pass the <your_ethernet_frame_object_name> and the output_port to the self.resend_packet() method
        # this will send the message to the router.
        self.resend_packet( ether1, output_port)
        log.debug('SECOND SCENARIO')
```

### 4.2.3 Cenário 3

Neste cenário efetua-se uma alteração nos endereços MAC da ethernet frame recebida, a fonte é o endereço MAC da interface de saída do router e o endereço MAC destino é obtido através da tabela ARP com o endereço IP de destino do pacote inicialmente recebido.

Para o envio da frame a estratégia é igual ao cenário anterior.

```
if destination_ip in self.arp_table:
    log.debug("THIRD SCENARIO")
    # here you must use the incoming etherFrame and change the
    etherFrame.src = mac_addr_inter
    # and change the 'dst' field to by the mac of the 'destination_ip'
    etherFrame.dst = EthAddr(self.arp_table[destination_ip])
    # you then send the frame by uncomment the following line
    self.resend_packet(etherFrame, output_port)
```

### 4.3 Testes

Fazendo um ping do host 1 para um interface do router.

```
"Node: h1"
root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code# ping -c1 10.0.3.1
PING 10.0.3.1 (10.0.3.1) 56(84) bytes of data:
64 bytes from 10.0.3.1: icmp_seq=1 ttl=64 time=1.37 ms

--- 10.0.3.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.368/1.368/1.368/0.000 ms
```

```
DEBUG:vr_tp3_ex2:RECEIVED: EthernetType -> IP from 1
DEBUG:vr_tp3_ex2:PACKET IS ROUTABLE!
DEBUG:vr_tp3_ex2:ICMP ECHO REPLY SENT!
DEBUG:vr_tp3_ex2:FIRST SCENARIO
DEBUG:vr_tp3_ex2:RECEIVED: EthernetType -> ARP to aa:bb:cc:dd:ee:01
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 1
DEBUG:vr_tp3_ex2:IT'S AN ARP REQUEST!
Reques dst IP: 10.0.1.1
DEBUG:vr_tp3_ex2:ARP REPLY SENT! 1
```

Fazendo um pingo do host 1 para o host 3 com endereço IP 10.0.3.100.

```
"Node: h1"
root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code# ping -c1 10.0.3.100
PING 10.0.3.100 (10.0.3.100) 56(84) bytes of data:
64 bytes from 10.0.3.100: icmp_seq=1 ttl=64 time=49.0 ms

--- 10.0.3.100 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 49.017/49.017/49.017/0.000 ms
root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code#
```



```

DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> ARP to ff:ff:ff:ff:ff:ff
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 1
DEBUG:vr_tp3_ex2:IT'S AN ARP REQUEST!
Reques dst IP: 10.0.1.1
DEBUG:vr_tp3_ex2:ARP REPLY SENT! 1
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> IP from 1
DEBUG:vr_tp3_ex2:PACKET IS ROUTABLE!
DEBUG:vr_tp3_ex2:SECOND SCENARIO
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> ARP to aa:bb:cc:dd:ee:03
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 3
DEBUG:vr_tp3_ex2:IT'S AN ARP REPLY!
DEBUG:vr_tp3_ex2:10.0.3.100 00:00:00:00:00:03 INSTALLED TO CAM TABLE
Sending buffered ICMP!
10.0.3.0/24
DEBUG:vr_tp3_ex2:Flow mod for destination network 10.0.3.0/24 sent!
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> IP from 3
DEBUG:vr_tp3_ex2:PACKET IS ROUTABLE!
DEBUG:vr_tp3_ex2:SECOND SCENARIO
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> ARP to aa:bb:cc:dd:ee:01
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 1
DEBUG:vr_tp3_ex2:IT'S AN ARP REPLY!
DEBUG:vr_tp3_ex2:10.0.1.100 00:00:00:00:00:01 INSTALLED TO CAM TABLE
Sending buffered ICMP!
10.0.1.0/24
DEBUG:vr_tp3_ex2:Flow mod for destination network 10.0.1.0/24 sent!
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> ARP to aa:bb:cc:dd:ee:03
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 3
DEBUG:vr_tp3_ex2:IT'S AN ARP REQUEST!
Reques dst IP: 10.0.3.1
DEBUG:vr_tp3_ex2:ARP REPLY SENT! 3

```

Voltando a fazer o mesmo ping novamente o resultado é diferente pois já existe essa entrada na tabela de fluxo.

```

DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> ARP to aa:bb:cc:dd:ee:03
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 3
DEBUG:vr_tp3_ex2:IT'S AN ARP REQUEST!
Reques dst IP: 10.0.3.1
DEBUG:vr_tp3_ex2:ARP REPLY SENT! 3
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> ARP to aa:bb:cc:dd:ee:01
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 1
DEBUG:vr_tp3_ex2:IT'S AN ARP REQUEST!
Reques dst IP: 10.0.1.1
DEBUG:vr_tp3_ex2:ARP REPLY SENT! 1

```

Quando se faz um ping a um host que não existe.

```
"Node: h1"
root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code# ping -c1 10.0.5.100
PING 10.0.5.100 (10.0.5.100) 56(84) bytes of data.

--- 10.0.5.100 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

root@goncalo-VirtualBox:/home/goncalo/Desktop/tp3/code#
```

```
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> IP from 1
DEBUG:vr_tp3_ex2:PACKET IS NOT ROUTABLE!
DEBUG:vr_tp3_ex2:ICMP DESTINATION UNREACHABLE SENT
DEBUG:vr_tp3_ex2:RECEIVED: EtherType -> ARP to aa:bb:cc:dd:ee:01
DEBUG:vr_tp3_ex2:ARP FRAME RECEIVED FROM 1
DEBUG:vr_tp3_ex2:IT'S AN ARP REQUEST!
Reques dst IP: 10.0.1.1
DEBUG:vr_tp3_ex2:ARP REPLY SENT! 1
```

## 5 Conclusão

Após a realização deste trabalho prático o grupo adquiriu novos conhecimentos e sobretudo aprofundou e consolidou o que já sabíamos da parte teórica da unidade curricular. Como é costume, ao colocar em prática a matéria lecionada na parte teórica, surgem várias dúvidas sobre as quais é preciso efetuar pesquisas para melhor compreensão sobre como realmente funciona este protocolo.

O grupo conclui que os objetivos foram alcançados e realça a grande vantagem da utilização deste protocolo quando se trata da manipulação das tabelas de fluxo no switch por parte do controlador tendo sido também essencial a utilização da API POX que facilitou todo este processo.

Devido á falta de tempo disponível por parte dos elementos não foi possível realizar a parte opcional do trabalho, sendo que após uma apreciação breve do problema, de facto, seria mais uma oportunidade para colocar á prova os nossos conhecimentos sobre esta área.