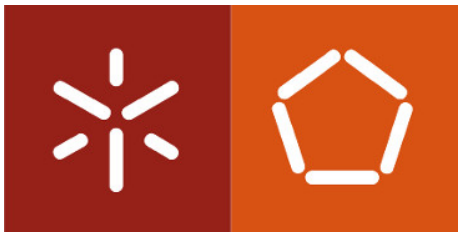
Virtualização de Redes

17 de Junho de 2021

a83899	André Morais
A85367	Francisco Lopes
A84485	Tiago Magalhães

OpenFlow



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

Resumo

Neste relatório, além de respondermos às questões presentes no enunciado, explicamos a nossa abordagem de a concluir ambos os exercícios propostos pelos trabalho prático. No primeiro exercício do trabalho foi construído, a partir do código fornecido, um controller OpenFlow que funciona como um switch de layer 2. No segundo execícios foi-nos proposta implementar um forwarder/switch estático de layer 3, para o efeito utilizamos dois ficheiros, com código que deveremos complementar, de forma a atingir o resultado pedido. uma construção uma rede de suporte baseada em nomes e tolerante a atrasos.

Keywords: Layer2, Layer3, Switch, OpenFlow

1 Introdução

No âmbito da Unidade Curricular de Virtualização de Redes, foi-nos proposta num primeiro exercício a construção de MAC-learning Switch de layer 2 e no exercício seguinte a implementação de um forwarder/switch de layer 3.

2 Questões

2.1 Q1 - What type of OpenFlow message is used to retrieve this information?

O comando executado imprime para a consola todas as informações de um switch, incluindo a informação nas flows tables do mesmo. Para tal o controlador envia mensagem OFPT_FEATURES_REQUEST e por sua vez recebe OFPT_FEATURES_REPLY

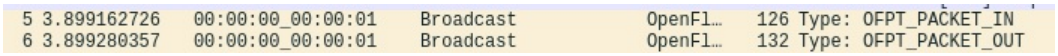
2.2 Q2 - Write the ovs-ofctl command to delete just the first Flow Table entry. Write the ovs-ofctl command to delete all entries of s1.

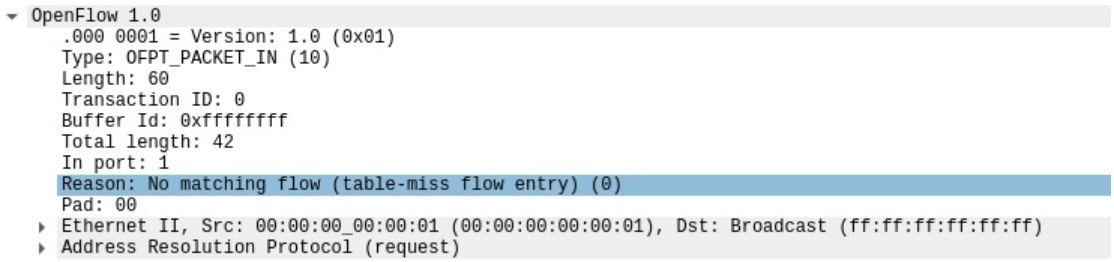
Para apagar a primeira entrada da Flow Table utilizamos *ovs-ofctl [–bundle] [–strict] del-flows switch [flow]* Para apagar todas as entradas utilizamos *ovs-ofctl [–bundle] del-flows switch*

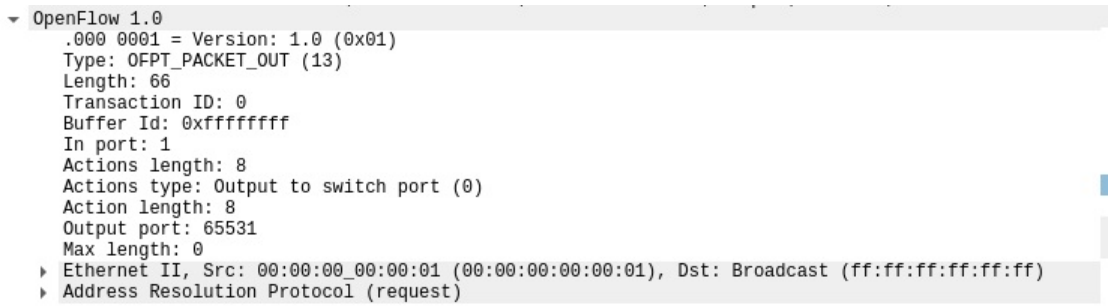
2.3 Q3 - Describe the start up communication between switch and controller, draw a sequence diagram that shows the type of OpenFlow messages being exchanged. Explain their purpose. Present a print of the Wireshark capture.

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

Com base nas imagens que demonstram dois pacotes, entre vários, capturados pelo wireshark, podemos afirmar que quando o comando é executado o switch recebe um pedido para o qual ainda não sabe o flow, então vai questionar o controlador o que fazer. Como é a primeira vez que o controlador está a lidar com o pedido vai ordenar que o switch envie para todas as portas o pedido de h1 exceto para h1.







2.5 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?

O mesmo não acontece pois a partir do momento em que o o controlador aprende o que fazer com os pacotes que recebe com certo destino IP, já sabe para qual interface enviar, pois cria uma flow entry na flow table. Aí o switch já nem precisa de questionar o controlador o que fazer pois já aprendeu como agir. Por esse motivo não vemos OFPT_PACKET_IN, nem OFPT_PACKET_OUT.

3 Exercício 1

O objetivo deste exercício é implementar um MAC-learning switch de layer 2, de forma a atingir o mesmo, deveremos complementar e modificar o ficheiro fornecido pelo professor de um controlador estilo hub. Essencialmente o hub recebe um packet e envia a toda a gente, o contrário acontece com o switch, verifica se tem o MAC guardado em memória e se o mesmo tiver envia para a porta correspondente, caso não se verifique envia para todas as portas.

3.1 Implementação

No código fornecido já existe um dicionário declarado para guardar os macs **mac_to_port**. Desta forma a primeira abordagem a executar quando entra um packet no switch é verificar se o MAC do enviador consta no dicionário, se o mesmo não acontecer, guardamo-lo-emos para possível uso futuro, associando a porta.

```
if str(in_mac) not in self.mac_to_port:
  print('Adicionei -> ' + str(in_mac) + ' porta ' + str(in_port))
  self.mac_to_port[str(in_mac)] = in_port
```

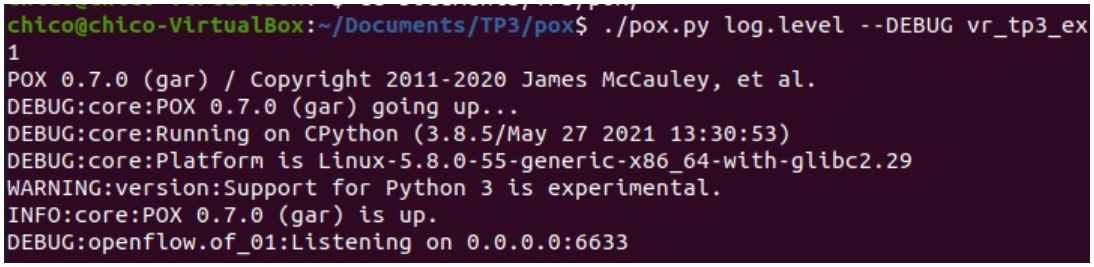
Após isso criamos um objeto **ofp_match**, assim podemos obter o endereço MAC de destino. De seguida existem dois cenários que podem acontecer, o endereço MAC de destino já estar registado e se assim acontecer vamos enviar o packet para a porta associada, utilizando o método **resend_create_flow(self,packet_in,out_port)**, se tal não acontecer o packet é enviado para todas as portas utilizando como porta **of.OFPP ALL**.

O método **resend_create_flow(self,packet_in,out_port)** foi implementado pelo grupo de forma a limpar um bocado o código ficando de mais fácil compreensão, de uma forma simples o que este método faz é cria um ofp_flow_mod adicionando um ofp_action_outport ao mesmo antes de enviar o pacote para a porta destino.

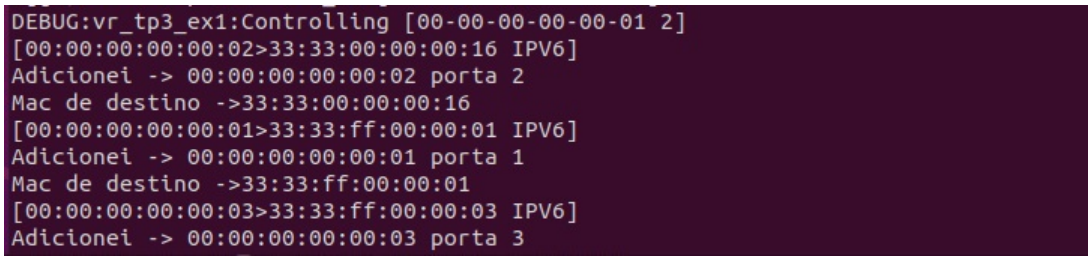
```
def resend_create_flow(self,packet_in,out_port):
ofp_flow_mod = of.ofp_flow_mod()
ofp_flow_mod.actions.append(of.ofp_action_output(port = out_port))
self.resend_packet(packet_in,out_port)
```

3.2 Testes

Antes de iniciar a topologia com recurso ao mininet é necessário iniciar o POX controller, configurando-o para utilizar o código desenvolvido. Com recurso ao comando *./pox.py log.level –DEBUG vr_tp3_ex1.*



Após vamos iniciar o mininet com o comando *sudo mn –topo single,3 –mac –switch ovsk –controller remote* , enquando observar-mos o terminal com o POX. Verificamos que existem pacotes a circular e que o mesmo está a registar os endereços MAC que não tem em memória.



De seguida vamos fazer efetuar um ping do host host 1 para o host 2 para comprovar que os hosts comunicam entre si.

```
mininet> h1 ping h2 -c3
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=64 time=10.8 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=2.34 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=4.50 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2130ms
rtt min/avg/max/mdev = 2.342/5.893/10.834/3.603 ms
```

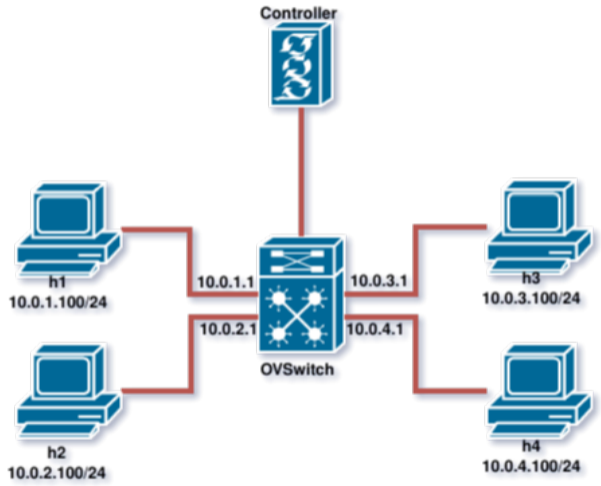
4 Exercício 2

No exercício 2 é proposto implementar um fowarding/switch de layer 3, semelhante ao exercício 1 a resolução passa por acrescentar código.

4.1 Implementação

4.1.1 Topologia

No enunciado é apresentada uma imagem com uma topologia que é necessário recriar, completando o ficheiro **base_topo_ex2.py**. Será necessário acrescentar então os 3 hosts que



faltam, assinalando a sua default router, bem como o nome. E adicionar a ligação entre os hosts e o switch.

```
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 )
```

4.2 Fowarding/Switch

Após leitura atenta do enunciado ficamos a saber que existem três possíveis cenários, para cada explicaremos a abordagem.

4.2.1 Cenário 1

O primeiro cenário ocorre quando o endereço IP de destino é igual a um dos endereços IP das interfaces presentes no router. Para o efeito utilizamos um if que procura na routing_table, da seguinte forma:

```
if(self.routing_table[str(destination_network)]['RouterInterface'] == destination_ip):
```

Se a condição for verdadeira é um caso de reencaminha o pacote para o método **ICMP_Handler**.

4.2.2 Cenário 2

Este cenário ocorre quando o router não conhece o endereço MAC do IP destino, para confirmar se tal acontece, utilizarmo-emos a arp_table. O segundo passo será guardar o pacote para poder ser enviado após o ARP Reply chegar.

```
self.buffer[destination_ip] = {'IP_Packet': ip_packet,
                    'DestinationNetwork': destination_network}
```

De seguida iremos construir um objeto arp().

```
arpRequest = arp()
arpRequest.opcode = arp.REQUEST
arpRequest.protosrc =
IPAddr(self.routing_table[str(destination_network)]['RouterInterface'])
arpRequest.protodst = IPAddr(destination_ip)
```

Ao objeto arp fomos definindo alguns paramêtros de acordo com os steps do enunciado.

```
arpRequest.hwsrc =
EthAddr(self.arp_table[self.routing_table[destination_network]['RouterInterface']])
arpRequest.hwdst = EthAddr('00:00:00:00:00:00')
```

Para finalizar antes de enviar o ARP REQUEST é necessário encapsula-lo dentro de um Ethernet Frame, para tal vamos criar um objeto ethernet() e definir alguns valores necessários. E após enviar utilizando o método **resend_packet**.

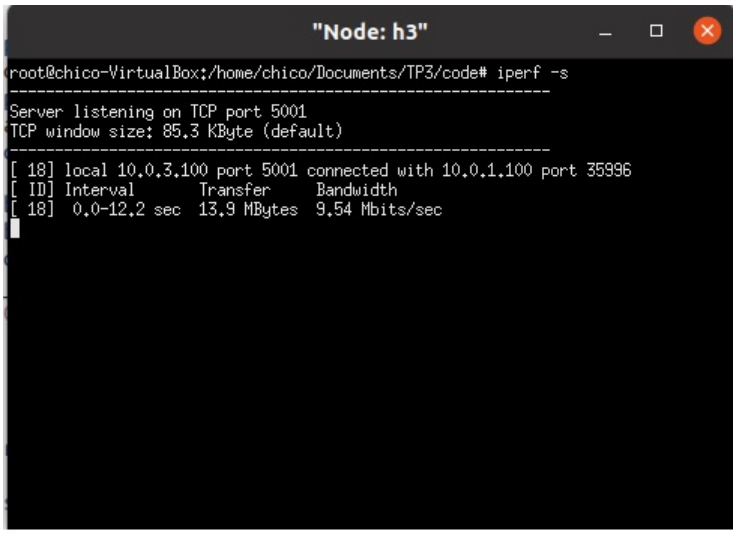
```
ether = ethernet()
ether.type = ethernet.ARP_TYPE
ether.src =
EthAddr(self.arp_table[self.routing_table[destination_network]['RouterInterface']])
ether.dst = EthAddr('FF:FF:FF:FF:FF:FF')
ether.payload = arpRequest
self.resend_packet(ether, output_port)
```

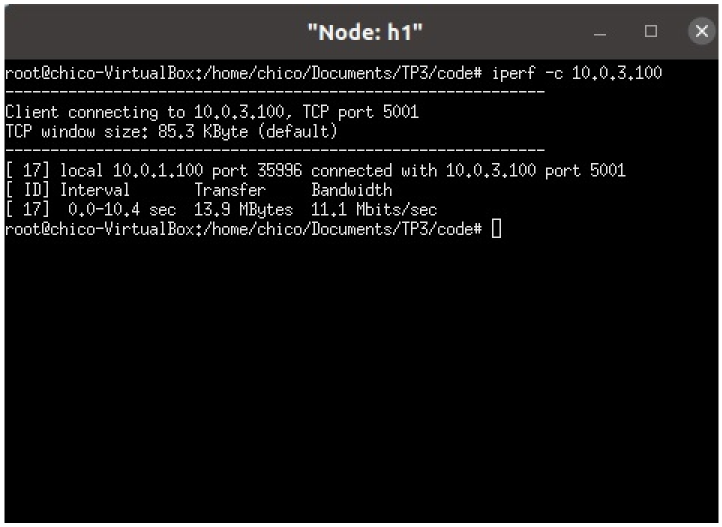
4.2.3 Cenário 3

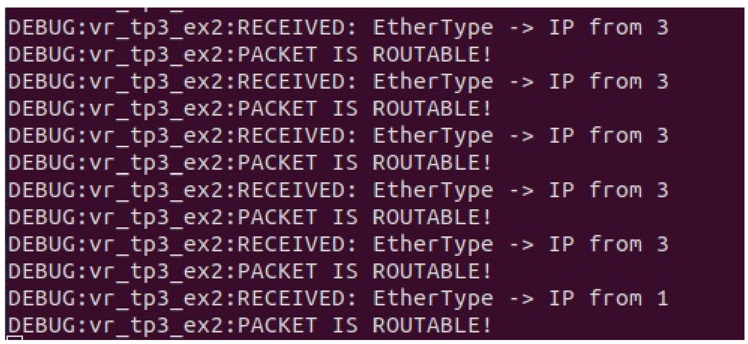
O cenário 3 ocorre quando o cenário 2 não se verifica. Quando este aconte simplesmente é necessário modificar os campos do etherframe recebido e enviar o pacote.

```
etherFrame.src =
EthAddr(self.arp_table[self.routing_table[destination_network]['RouterInterface']])
etherFrame.dst = EthAddr(self.arp_table[destination_ip])
self.resend_packet(etherFrame, output_port)
```

4.3 Testes







5 Conclusão

Finalizando, os exercícios proposto foram resolvidos, conseguindo assim atingir o objetivo do trabalho que seria familiarizar os alunos com OpenFlow. Verificamos também que é possível, usando o mesmo hardware, realizar diferentes cenários utilizando OpenFlow e código relativamente simples.