



TÉCNICO
LISBOA

Engenharia de Tráfego
2018-2019

Mestrado em Engenharia Telecomunicações e Informática

Relatório

Software-Defined Networking and OpenFlow

Introduction to SDN controllers

domingo, 28 de novembro de 2018

Grupo 5

João Correia - 81990
João Costa - 82528

Introdução

Este laboratório tem como objetivo complementar o anterior, metendo em prática o uso de um controlador, usando a plataforma POX. Esta plataforma oferece uma *Framework* que possibilita a comunicação entre o gestor de rede e os dispositivos de encaminhamento de pacotes compatíveis com o protocolo OpenFlow (*switches* e *routers*).

É também estudado o comportamento de um *hub* e de um *switch*. Numa parte mais avançada do laboratório é estudada a eficiência destes dois elementos, analisando os *delays* dos pacotes pela rede através da ferramenta ***ping***.

No final é alterado o comportamento do *switch* de modo a que este aprenda automaticamente os fluxos necessários para o encaminhamento de pacotes.

Inicialização

Começamos por iniciar o Mininet numa máquina virtual (usando a imagem Mininet VM). Para nos ligarmos ao servidor usamos SSH, tendo sido descoberto anteriormente o endereço IP da máquina Mininet através do comando **ifconfig**. O comando SSH é efetuado com a **flag -Y** para que seja possível visualizar graficamente os comandos enviados para o servidor (onde estes são corridos).

Foi criada uma topologia igual ao do laboratório passado, com 3 *hosts*, 1 *switch* e 1 controlador (comando: **sudo mn --topo single,3 --mac --switch ovsk --controller remote**). De seguida foi aberto um novo terminal e executado o POX através do ficheiro *script pox.py* existente na diretoria ~/pox. Foram introduzidas algumas *flags* no comando para fazer *debug* ao mesmo e iniciar o componente *of_tutorial* (comando: **./pox.py log.level --DEBUG misc.of_tutorial**).

Neste exercício, o controlador irá agir como um *hub*, ou seja, redirecionará todos os pacotes para todas as interfaces do *switch*, exceto a de entrada do pacote. O POX só funcionará corretamente se for lançado antes da topologia. Na Figura 1 podemos ver o POX conectado ao *switch*.

```
2. mininet@mininet-vm: ~/pox (ssh)
mininet@mininet-vm:~/pox$ ./pox.py log.level --DEBUG misc.of_tutorial
POX 0.2.0 (carp) / Copyright 2011-2013 James McCauley, et al.
DEBUG:core:POX 0.2.0 (carp) going up...
DEBUG:core:Running on CPython (2.7.6/0ct 26 2016 20:30:19)
DEBUG:core:Platform is Linux-4.2.0-27-generic-x86_64-with-Ubuntu-14.04-trusty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[None 1] closed
INFO:openflow.of_01:[00-00-00-00-00-01 2] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 2]
```

Figure 1 - Inicialização do POX

Comportamento do hub

Para testar o comportamento do controlador, foi efetuado um *ping* do *h1* para o *h2*. Uma vez que o *switch* tem o comportamento de um *hub*, é de esperar que o *h3* também recebe o pacote enviado pelo *h1* para o *h2*. Podemos ver este comportamento na Figura 2, sendo que a parte de cima da figura é referente ao *h3* e a de baixo ao *h2*.

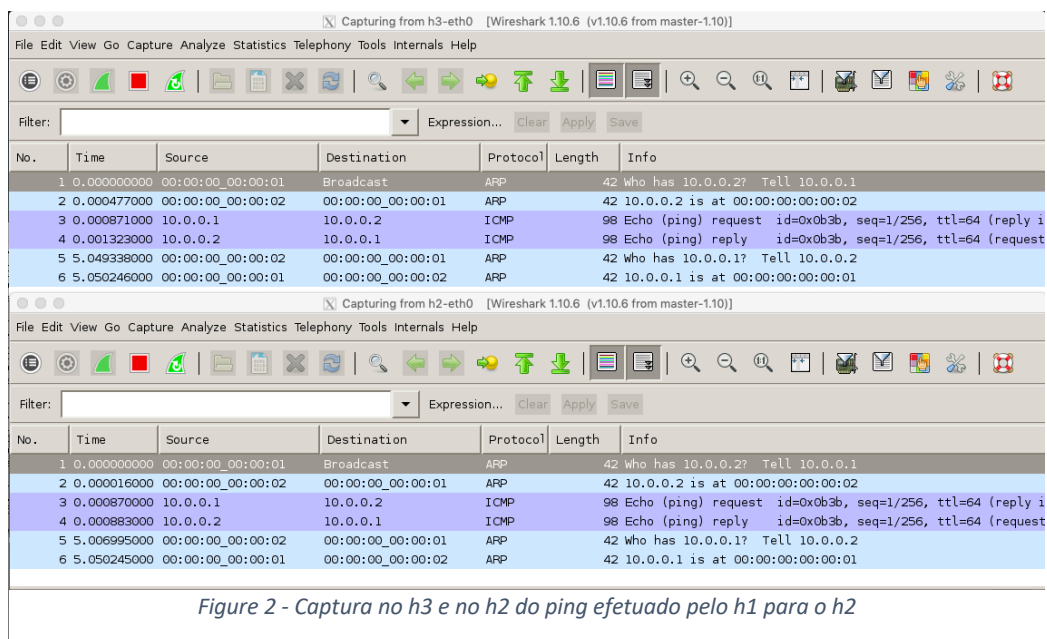


Figure 2 - Captura no h3 e no h2 do ping efetuado pelo h1 para o h2

Em relação às mensagens trocadas, como o *h1* não possui o *MAC Address* do *h2*, envia uma mensagem em *broadcast* para a perguntar quem é o *h2*. Este quando recebe esta primeira mensagem ARP, responde com o seu *MAC Address*. A partir deste momento, ambos os nós passam a ter conhecimento um do outro, podendo assim efetuar o *ping* ICMP (terceira e quarta mensagem). Apesar do *h2* já ter conhecimento do *MAC Address* do *h1*, este envia uma mensagem direcionada ao *h1* para confirmar o seu endereço IP, obtendo de seguida a resposta à mensagem ARP.

Se for efetuado um *ping* para um endereço IP que não esteja a ser usado, o *host* que envia a mensagem não obterá qualquer resposta (ping do *h1* para o endereço IP 10.0.0.5). Podemos verificar este acontecimento na Figura 3.

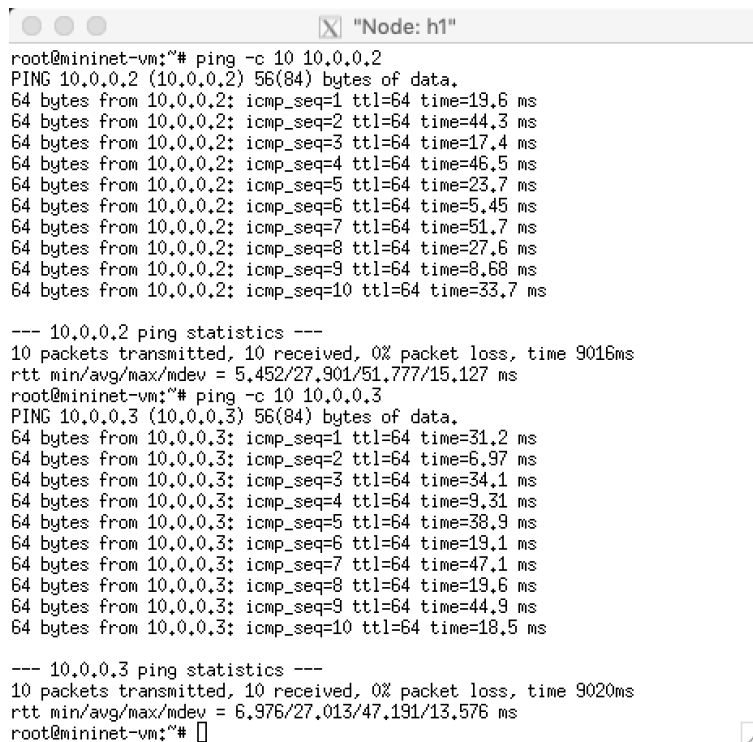
The figure consists of two Wireshark packet capture screenshots. The top screenshot is titled 'Capturing from h3-eth0 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]' and shows a list of 9 packets. The bottom screenshot is titled 'Capturing from h2-eth0 [Wireshark 1.10.6 (v1.10.6 from master-1.10)]' and shows a list of 9 packets. Both screenshots show a mix of ARP and ICMP traffic.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	00:00:00_00:00:01	Broadcast	ARP	42	who has 10.0.0.2? Tell 10.0.0.1
2	0.000477000	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	10.0.0.2 is at 00:00:00:00:00:02
3	0.000871000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0b3b, seq=1/256, ttl=64 (reply i
4	0.001323000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0b3b, seq=1/256, ttl=64 (request
5	5.049338000	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	who has 10.0.0.1? Tell 10.0.0.2
6	5.050246000	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01
7	601.104726000	00:00:00_00:00:01	Broadcast	ARP	42	who has 10.0.0.5? Tell 10.0.0.1
8	602.084530000	00:00:00_00:00:01	Broadcast	ARP	42	who has 10.0.0.5? Tell 10.0.0.1
9	603.112389000	00:00:00_00:00:01	Broadcast	ARP	42	who has 10.0.0.5? Tell 10.0.0.1

No.	Time	Source	Destination	Protocol	Length	Info
3	0.000870000	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x0b3b, seq=1/256, ttl=64 (reply
4	0.000883000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x0b3b, seq=1/256, ttl=64 (request
5	5.006995000	00:00:00_00:00:02	00:00:00_00:00:01	ARP	42	who has 10.0.0.1? Tell 10.0.0.2
6	5.050245000	00:00:00_00:00:01	00:00:00_00:00:02	ARP	42	10.0.0.1 is at 00:00:00:00:00:01
7	601.104725000	00:00:00_00:00:01	Broadcast	ARP	42	who has 10.0.0.5? Tell 10.0.0.1
8	602.084529000	00:00:00_00:00:01	Broadcast	ARP	42	who has 10.0.0.5? Tell 10.0.0.1
9	603.112388000	00:00:00_00:00:01	Broadcast	ARP	42	who has 10.0.0.5? Tell 10.0.0.1

Figure 3 - Ping *h1* para o endereço IP 10.0.0.5 (inexistente)

Ao efetuarmos dois *pings* distintos, ambos provenientes do *h1* e como origem *h2* e *h3* respetivamente, podemos verificar que não há uma grande variação entre os dois resultados, tanto para o RTT mínimo, máximo como para o RTT médio. Podemos visualizar este comportamento na Figura 4.



```
root@mininet-vm:~# ping -c 10 10.0.0.2
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=19.6 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=44.3 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=17.4 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=46.5 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=23.7 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=5.45 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=51.7 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=27.6 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=8.68 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=33.7 ms

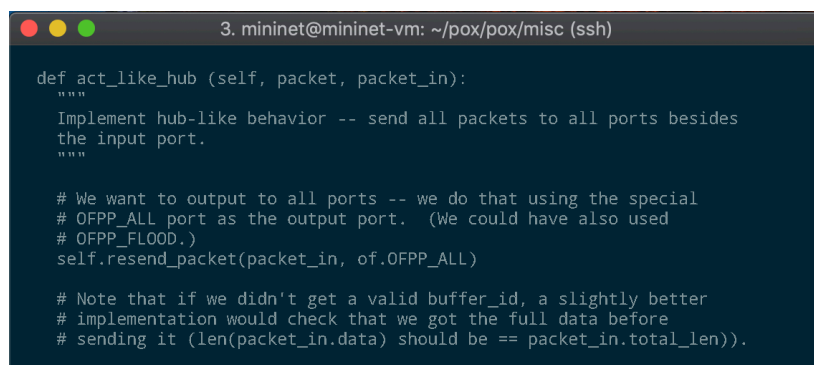
--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9016ms
rtt min/avg/max/mdev = 5.452/27.901/51.777/15.127 ms
root@mininet-vm:~# ping -c 10 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=31.2 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=6.97 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=34.1 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=9.31 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=38.9 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=19.1 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=47.1 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=19.6 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=44.9 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=18.5 ms

--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9020ms
rtt min/avg/max/mdev = 6.976/27.013/47.191/13.576 ms
root@mininet-vm:~#
```

Figure 4 - Ping do h1 para o h2 e de seguida do h1 para o h3

POX of_tutorial code for simple hub

Após analisar o código, conseguimos perceber que o comportamento do *switch* é dado pela função **act_like_hub()**, código este que está presente na Figura 5.



```
3. mininet@mininet-vm: ~/pox/pox/misc (ssh)

def act_like_hub (self, packet, packet_in):
    """
    Implement hub-like behavior -- send all packets to all ports besides
    the input port.
    """

    # 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.)
    self.resend_packet(packet_in, of.OFPP_ALL)

    # Note that if we didn't get a valid buffer_id, a slightly better
    # implementation would check that we got the full data before
    # sending it (len(packet_in.data) should be == packet_in.total_len)).
```

Figure 5 - Comportamento do switch definido pela função **act_like_hub**

Esta função recebe como argumentos dois tipos de pacote, um primeiro (**packet**) cujo conteúdo é MAC Address da origem e do destino, e o protocolo usado, e um segundo (**packet_in**) que contém informação relacionada com o pacote OpenFlow direcionada ao controlador. Estas mensagens são visíveis na Figura 6 (foi necessário alterar o ficheiro *of_tutorial.py*).

```
2. mininet@mininet-vm: ~/pox (ssh)
PACKET
[00:00:00:00:02>00:00:00:00:00:01 ARP]
PACKET_IN
ofp_packet_in
header:
  version: 1
  type: 10 (OFPT_PACKET_IN)
  length: 60
  xid: 0
  buffer_id: 257
  total_len: 42
  in_port: 2
  reason: 0
  data:

PACKET
[00:00:00:00:01>00:00:00:00:00:02 IP]
PACKET_IN
ofp_packet_in
header:
  version: 1
  type: 10 (OFPT_PACKET_IN)
  length: 116
  xid: 0
  buffer_id: 258
  total_len: 98
  in_port: 1
  reason: 0
  data: ETuh@@@>
! "#$%&'()*+,-./01234567
```

Figure 6 - Conteúdo das variáveis *packet* e *packet_in*

A função limita-se a avisar o *switch* para que este reenvie o pacote que recebeu do *h1* para todas as portas (exceto a de entrada do pacote). Esta instrução é dada através da função ***resend_packet()***, tendo como argumento a variável *packet_in* (usada para avisar o controlador que não sabe para onde enviar o pacote recebido) e a variável ***of.OFPP_ALL*** que corresponde a todas as portas do *switch*, sendo que serão usadas como portas de saída.

Implementing a learning switch

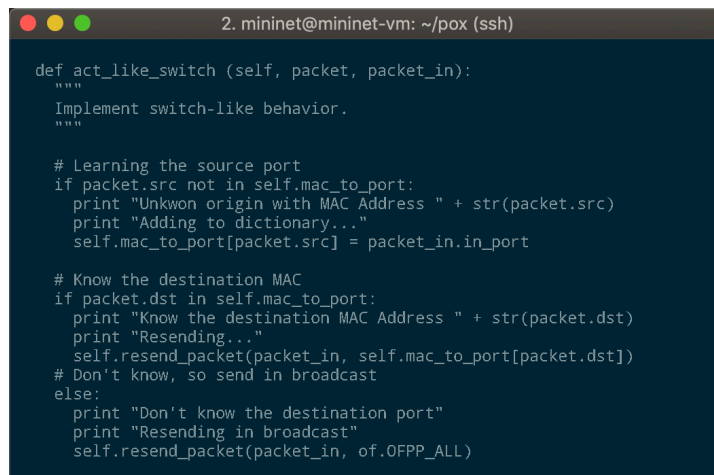
De maneira a alterar o comportamento do *switch* para que não enviasse os pacotes para todas as suas interfaces (deixasse de se comportar como um *hub*), foi alterado o ficheiro ***of_tutorial.py***.

Primeiro foi comentada a linha ***self.act_like_hub(packet, packet_in)*** e descomentada a linha ***self.act_like_switch(packet, packet_in)*** da função ***_handle_PacketIn()***. De seguida completou-se a função ***act_like_switch()***.

Nesta função é verificado se o *MAC Address* do pacote de origem está presente no dicionário (***mac_to_port***) que relaciona os *MAC Addresses* dos *hosts* com as portas do *switch*. Se não tiver é adicionado.

De seguida, e para reencaminhar o pacote para o destino correto, é verificado se o *MAC Address* do destino está presente no dicionário ***mac_to_port***. Se estiver o pacote é preparado na função ***resend_packet()***, recebendo como argumentos o pacote OpenFlow e a porta para a qual o *switch* tem que enviar o pacote recebido. Se não tiver o *MAC Address* no dicionário, o controlador irá dar a instrução ao *switch* para que este envie através de todas as suas portas (menos a de chegada do pacote).

Na Figura 7 podemos ver as alterações efetuadas na função ***act_like_switch()***.



```

2. mininet@mininet-vm: ~/pox (ssh)

def act_like_switch (self, packet, packet_in):
    """
    Implement switch-like behavior.
    """

    # Learning the source port
    if packet.src not in self.mac_to_port:
        print "Unkwn origin with MAC Address " + str(packet.src)
        print "Adding to dictionary..."
        self.mac_to_port[packet.src] = packet_in.in_port

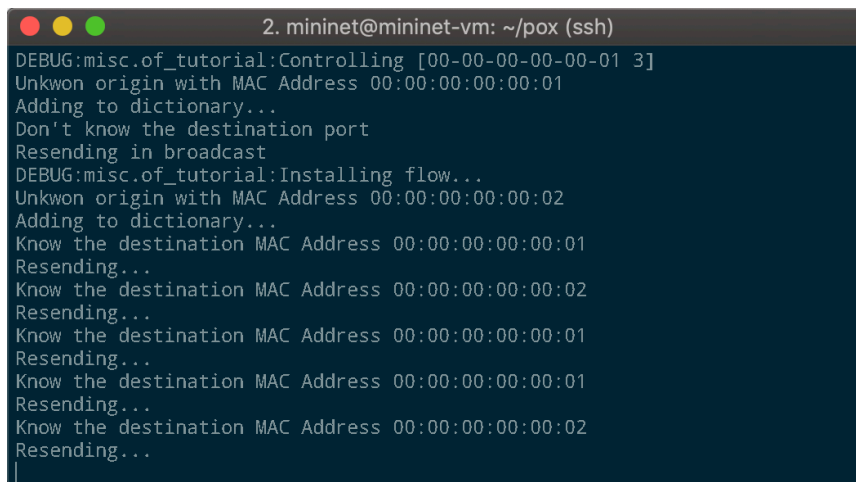
    # Know the destination MAC
    if packet.dst in self.mac_to_port:
        print "Know the destination MAC Address " + str(packet.dst)
        print "Resending..."
        self.resend_packet(packet_in, self.mac_to_port[packet.dst])
    # Don't know, so send in broadcast
    else:
        print "Don't know the destination port"
        print "Resending in broadcast"
        self.resend_packet(packet_in, of.OFPP_ALL)

```

Figure 7 - Alterações feitas à função `act_like_switch`

Para testar o funcionamento das alterações, foi efetuado um ping do *h1* para o *h2*. No terminal do POX foram registadas as seguintes mensagens, podendo também ser vistas na Figura 8:

- **Unkwn origin with MAC Address 00:00:00:00:00:01**
Como não conhece o *MAC Address* de origem, adiciona-o ao dicionário (mensagem em baixo)
- **Don't know the destination port**
Como não conhece o *destination port*, reenvia a mensagem em *broadcast*.
- **Know the destination MAC Address 00:00:00:00:00:01**
Como conhece o *MAC Address* de destino, reenvia a mensagem pela porta associada ao mesmo



```

2. mininet@mininet-vm: ~/pox (ssh)
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 3]
Unkwn origin with MAC Address 00:00:00:00:00:01
Adding to dictionary...
Don't know the destination port
Resending in broadcast
DEBUG:misc.of_tutorial:Installing flow...
Unkwn origin with MAC Address 00:00:00:00:00:02
Adding to dictionary...
Know the destination MAC Address 00:00:00:00:00:01
Resending...
Know the destination MAC Address 00:00:00:00:00:02
Resending...
Know the destination MAC Address 00:00:00:00:00:01
Resending...
Know the destination MAC Address 00:00:00:00:00:01
Resending...
Know the destination MAC Address 00:00:00:00:00:02
Resending...

```

Figure 8 - Comportamento do controlador após receber mensagens do switch

É também possível verificar, na Figura 9, que o *host h3* só recebe a primeira mensagem, enviada em *broadcast*, e o *h2* recebe todas as mensagens. É assim possível concluir que o *switch* já não se comporta como um *hub*.

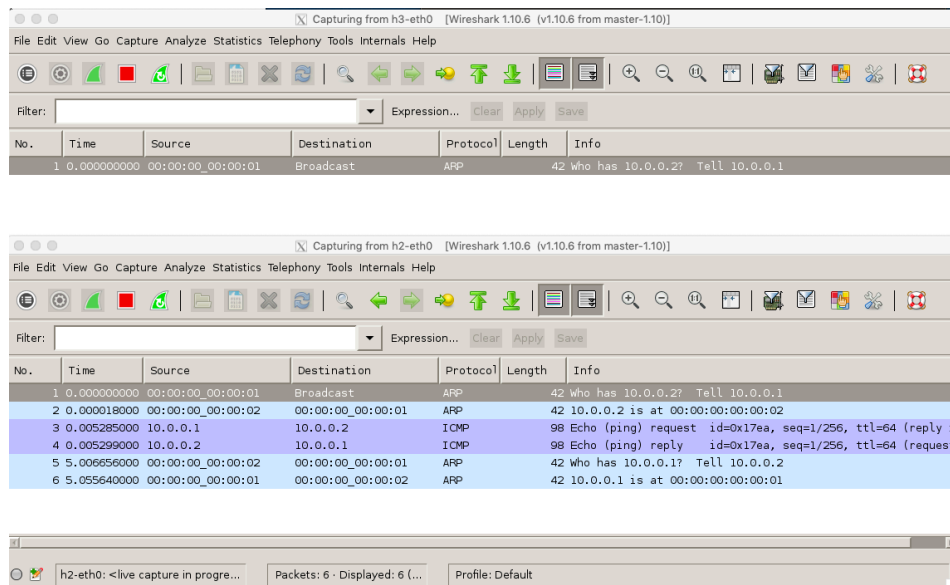


Figure 9 - Captura do ping do h1 para o h2, no h3 e h2 respetivamente

Em termos de *performance* não houve melhorias no RTT, devido ao facto dos pacotes terem a necessidade de ir ao controlador, mesmo que já se saiba qual a porta do *switch* por onde este tem de reenviar. Podemos ver estes novos resultados na Figura 10

```

root@mininet-vm:~# ping -c 10 10.0.0.2
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=5.82 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=38.8 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=18.6 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=49.5 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=31.8 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=56.3 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=52.6 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=33.9 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=9.96 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=39.8 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9017ms
rtt min/avg/max/mdev = 5.828/33.756/56.374/16.614 ms
root@mininet-vm:~# ping -c 10 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=19.9 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=50.6 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=30.9 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=7.67 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=35.9 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=18.0 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=3.17 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=32.2 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=16.6 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=50.8 ms

--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9016ms
rtt min/avg/max/mdev = 3.170/26.617/50.884/15.581 ms
root@mininet-vm:~#

```

Figure 10 - Ping do h1 para o h2 e do h1 para o h3 respetivamente

Uma maneira de evitar que as mensagens tenham que ir constantemente ao controlador para o *switch* saber o que fazer com elas, o controlador pode atribuir regras de *flows* ao *switch*, para que este saiba automaticamente o que fazer com os pacotes que recebe.

Para implementar foi alterado novamente o código da função **act_like_switch()** do ficheiro **of_tutorial.py**.

Estas modificações podem ser encontradas na Figura 11.


```

3. mininet@mininet-vm: ~/pox/pox/misc (ssh)

def act_like_switch (self, packet, packet_in):
    """
    Implement switch-like behavior.
    """

    # Learning the source port
    if packet.src not in self.mac_to_port:
        print "Unkwn origin with MAC Address " + str(packet.src)
        print "Adding to dictionary..."
        self.mac_to_port[packet.src] = packet_in.in_port

    # Know the destination MAC
    if packet.dst in self.mac_to_port:
        print "Know the destination MAC Address " + str(packet.dst)
        print "Resending message"
        self.resend_packet(packet_in, self.mac_to_port[packet.dst])

        print "Creating rule and sending to switch"
        message = of.ofp_flow_mod()
        message.match.dl_dst = packet.dst
        action = of.ofp_action_output(port = self.mac_to_port[packet.dst])
        message.actions.append(action)
        self.connection.send(message)

    # Don't know, so send in broadcast
    else:
        print "Don't know the destination port"
        print "Resending in broadcast"
        self.resend_packet(packet_in, of.OFPP_ALL)

```

Figure 11 - Alteração do função `act_like_switch` para que o switch use a tabela de flows

Para que o *switch* crie um fluxo entre dois *hosts*, o controlador tem que enviar uma regra a este, sendo que a regra é caracterizada pelo *MAC Address* de destino e a correspondente porta. Deste modo, quando o *switch* receber o um pacote, verifica na sua tabela de fluxos qual a porta de destino para onde tem que reenviar este pacote.

No código alterado na Figura 11, foi criada uma mensagem do tipo *flow* (**`message = of.ofp_flow_mod()`**) e associado um *MAC Address* de destino (**`message.match.dl_dst = packet.dst`**). De seguida, é associada uma porta de destino a uma ação (**`action = of.ofp_action_output(port = self.mac_to_port[packet.dst])`**), sendo no final a ação agregada à mensagem (**`message.actions.append(action)`**). Depois de formada a mensagem (regra criada para o *flow*), o controlador envia-a para o *switch*.

Esta regra é criada quando o *switch* pergunta ao controlador para quem é que tem de enviar a mensagem, tendo obrigatoriamente o controlador de ter conhecimento do *host* de destino (*MAC Address* e porta do *switch* que liga a esse *host*).

Para testar o correto funcionamento dos *flows* no *switch* foi executado um novo *ping* do *h1* para o *h2* e verificadas as tabelas de *flows* do *switch*, como também as mensagens enviadas do *switch* para o controlador. Podemos ver este comportamento nas Figuras 12 e 13 respetivamente.

```

3. mininet@mininet-vm: ~ (ssh)

mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=18.244s, table=0, n_packets=1, n_bytes=42, idle_age=13, dl_dst=00:00:00:00:00:02 actions=output:2
 cookie=0x0, duration=18.284s, table=0, n_packets=2, n_bytes=140, idle_age=13, dl_dst=00:00:00:00:00:01 actions=output:1
mininet@mininet-vm:~$ |

```

Figure 12 - Tabela de flows antes e depois do ping

```

2. mininet@mininet-vm: ~/pox (ssh)
sty
INFO:core:POX 0.2.0 (carp) is up.
DEBUG:openflow.of_01:Listening on 0.0.0.0:6633
INFO:openflow.of_01:[00-00-00-00-00-01 1] connected
DEBUG:misc.of_tutorial:Controlling [00-00-00-00-00-01 1]
Unkwn origin with MAC Address 00:00:00:00:00:01
Adding to dictionary...
Don't know the destination port
Resending in broadcast
DEBUG:misc.of_tutorial:Installing flow...
Unkwn origin with MAC Address 00:00:00:00:00:02
Adding to dictionary...
Know the destination MAC Address 00:00:00:00:00:01
Resending message
Creating rule and sending to switch
Know the destination MAC Address 00:00:00:00:00:02
Resending message
Creating rule and sending to switch

```

Figure 13 – Comportamento do controlador após receber mensagens do switch

Na Figura 13 podemos ver mensagens de tratamento do controlador após receber mensagens provenientes do *switch*. Existem X tipos diferentes:

- **Unkwn origin with MAC Address 00:00:00:00:00:01**
Como não conhece o *MAC Address* de origem, adiciona-o ao dicionário (mensagem em baixo)
- **Don't know the destination port**
Como não conhece o *destination port*, reenvia a mensagem em *broadcast*.
- **Know the destination MAC Address 00:00:00:00:00:01**
Como conhece o *MAC Address* de destino, reenvia a mensagem pela porta associada ao mesmo e cria uma regra para o *switch* saber o que fazer quando voltar a receber uma mensagem com este par de *hosts* (origem, destino).

Podemos verificar deste modo, que o número de mensagens que passam pelo controlador diminui. Após efetuado um segundo *ping*, não houve mensagens adicionais trocadas entre o *switch* e este, pois o *switch* já sabia o que fazer com estas.

Como era de esperar, fazendo um *ping* do *h1* para o *h2*, o *host h3* apenas receberá a primeira mensagem enviada (enviada pelo *h1* em *broadcast* por não conhecer o *MAC Address* de destino). Podemos ver este comportamento na Figura 14.

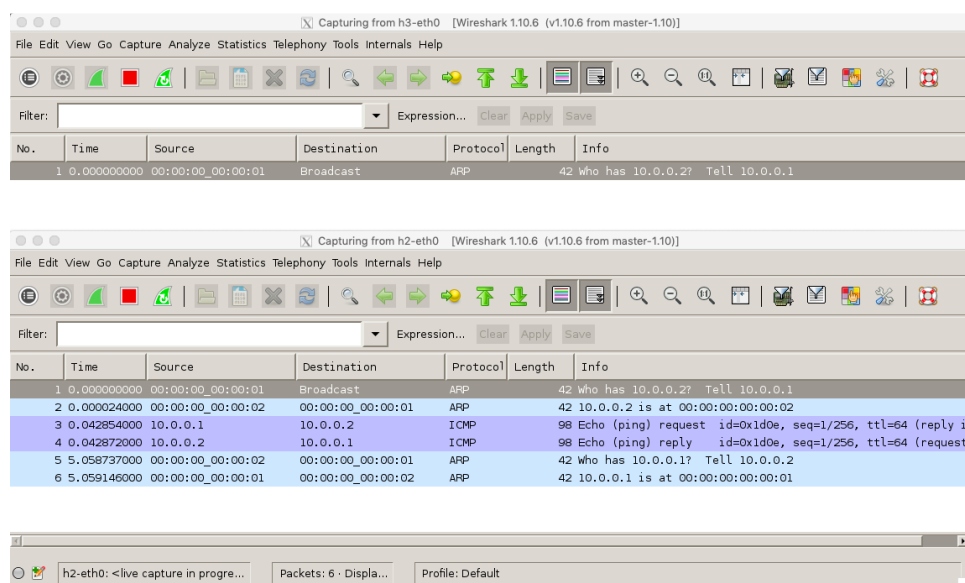


Figure 14 - Captura no h2 e h3 após um ping do h1 para o h2

Ao se fazer um novo ping, por exemplo do *h1* para o *h3*, é de esperar que a tabela de *flows* ganhe uma nova entrada, relativamente a *MAC Address* de destino do *host h3*. Podemos ver isto na Figura 15.

```

3. mininet@mininet-vm: ~ (ssh)
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=10.412s, table=0, n_packets=1, n_bytes=42, idle_age=5, dl_dst=00:00:00:00:00:02 actions=output:2
  cookie=0x0, duration=10.45s, table=0, n_packets=2, n_bytes=140, idle_age=5, dl_dst=00:00:00:00:00:01 actions=output:1
mininet@mininet-vm:~$ sudo ovs-ofctl dump-flows s1
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=33.338s, table=0, n_packets=1, n_bytes=42, idle_age=28, dl_dst=00:00:00:00:00:02 actions=output:2
  cookie=0x0, duration=17.626s, table=0, n_packets=1, n_bytes=42, idle_age=12, dl_dst=00:00:00:00:00:03 actions=output:3
  cookie=0x0, duration=33.376s, table=0, n_packets=5, n_bytes=322, idle_age=12, dl_dst=00:00:00:00:00:01 actions=output:1
mininet@mininet-vm:~$

```

Figure 15 - Entradas da tabela de flows após pings do *h1* para o *h2* e do *h1* para o *h3*

Com a implementação de *flows* no *switch*, podemos observar uma melhoria considerável no RTT máximo, mínimo e médio dos *pings* efetuados do *host h1* para o *h2* e *h3*, relativamente ao caso em que o *switch* não tinha entradas na tabela de *flows*.

Isto deve-se ao facto do *switch* não ter que ir ao controlador saber o que fazer com os pacotes que recebe, pois ao analisar a tabela sabe para que porta os tem que enviar.

Podemos ver esta melhor nos tempos RTT, na Figura 16.

```

Node: h1
root@mininet-vm:~$ ping -c 10 10.0.0.2
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=0.284 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.043 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.049 ms
64 bytes from 10.0.0.2: icmp_seq=6 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=7 ttl=64 time=0.041 ms
64 bytes from 10.0.0.2: icmp_seq=8 ttl=64 time=0.074 ms
64 bytes from 10.0.0.2: icmp_seq=9 ttl=64 time=0.044 ms
64 bytes from 10.0.0.2: icmp_seq=10 ttl=64 time=0.042 ms

--- 10.0.0.2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9000ms
rtt min/avg/max/ndev = 0.041/0.073/0.284/0.071 ms
root@mininet-vm:~$ ping -c 10 10.0.0.3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=0.365 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.055 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.051 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.040 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.049 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=0.051 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.050 ms
64 bytes from 10.0.0.3: icmp_seq=10 ttl=64 time=0.059 ms

--- 10.0.0.3 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8990ms
rtt min/avg/max/ndev = 0.037/0.080/0.365/0.095 ms
root@mininet-vm:~$

```

Figure 16 - Ping do *h1* para o *h2*, e *h1* para o *h3*

Comparação dos RTT efetuados pelo ping do *h1-h2* (sem tabela *flows* / com tabela de *flows*):

- RTT Máximo = 56.374 ms / 0.284 ms
- RTT Mínimo = 5.828 ms / 0.041 ms
- RTT Médio = 33.756 ms / 0.073 ms