



UMinho

**Mestrado Engenharia Informática
Redes Definidas por Software
(2022/2023)**

TP1

PG50334, Diogo Pires

PG50340, Diogo Matos

PG50723, Rita Gomes

Braga, March 19, 2023

Grupo 3

1 Introdução

A rede definida por software (SDN) é uma tecnologia que separa o plano de controlo de rede do plano de dados, fornecendo uma arquitetura de rede programável. O protocolo *OpenFlow* é um protocolo de comunicação SDN que permite que o controlador interaja com os dispositivos que estão encarregues pelo encaminhamento, tais como *switches* e *routers*.

O objetivo deste trabalho passa por desenvolver aplicações sobre controladores SDN, fazendo com que os alunos tenham um conhecimento mais alargado relativamente ao *OpenFlow*. Assim sendo, este trabalho baseia-se na resolução de dois exercícios, nos quais se pedia aos alunos que compreendessem e desenvolvessem aplicações que implementam um *switch* de *Layer-2* e uma topologia de autoaprendizagem na rede. O software utilizado para este trabalho inclui *Mininet*, *Ryu* e *Wireshark*.

De uma forma geral, o primeiro exercício fornece um tutorial para os alunos entenderem a aplicação que implementa um *switch* de autoaprendizagem de *Layer-2* de *OVSwitch*, enquanto que o segundo exercício requer que os alunos desenvolvam uma aplicação que monitorize os dispositivos *OVSwitch* e suas portas, links entre dispositivos, endereços IP dos *hosts* e as suas conexões para *OVSwitch* e, por fim, foi ainda pedido aos alunos que escrevessem ficheiros *YAML* que descrevessem a topologia.

2 Exercício 1

2.1 Estratégia adotada

Depois de instaladas todas as ferramentas pedidas, o grupo começou por implementar a topologia pedida no enunciado:

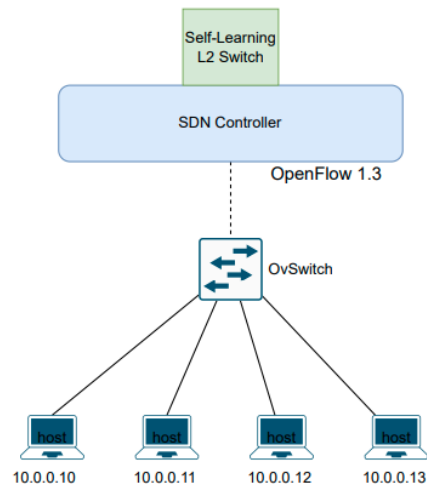


Figure 1: Topologia do exercício 1.

Na figura abaixo, podemos ver a topologia criada recorrendo ao *Mininet* e utilizando o comando `sudo mn -topo single,4 -controller=remote`.

```
*** Creating network
*** Adding controller
Connecting to remote controller at 127.0.0.1:6653
*** Adding hosts:
h1 h2 h3 h4
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1) (h3, s1) (h4, s1)
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
```

Figure 2: Topologia relativa ao exercício 1 implementada no *Mininet*.

2.2 Descrição da lógica da aplicação

Para uma melhor compreensão de todo o funcionamento da aplicação, o grupo esboçou um *flowchart* que mostra todo o processo depois de recebido um *packet.in*.

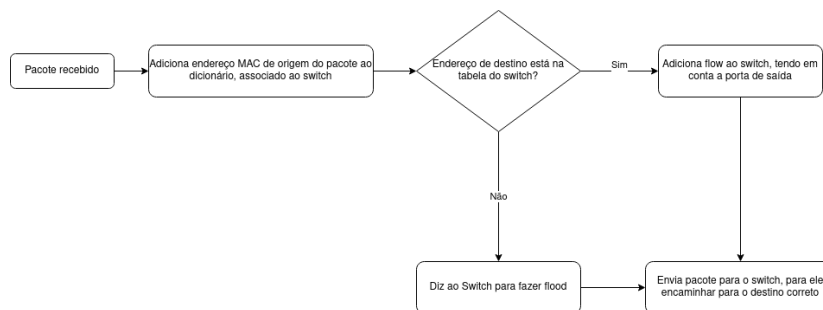


Figure 3: *Flowchart* depois da receção de um *packet.in*.

Em anexo, apresenta-se o código descrito a um nível mais detalhado, através de comentários.

2.2.1 Protocolo *OpenFlow*

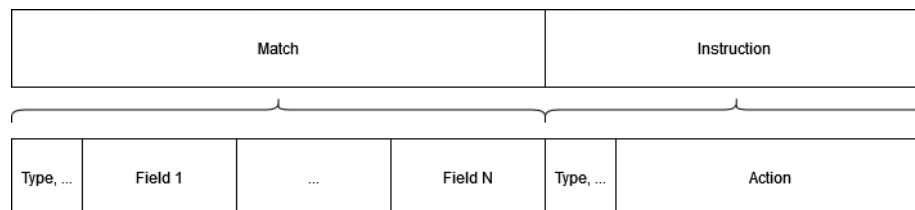


Figure 4: Estrutura de um pacote genérico

De seguida, apresentamos um exemplo de uma mensagem *OpenFlow*, contendo uma instrução para um *switch*:

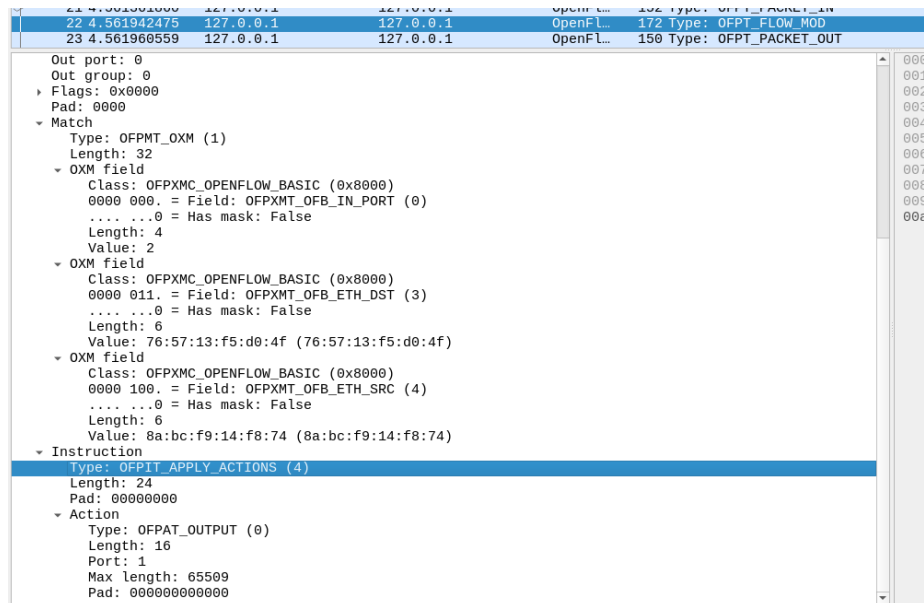


Figure 5: Exemplo retirado do *Wireshark*.

Como podemos ver, o *match* que o *switch* deve aplicar possui os campos de origem e destino de nível 2, e caso uma mensagem corresponda a esta descrição, deve ser reencaminhada pela porta 1.

De seguida, apresentamos os *FlowMods* existentes no código:

```
# Outro exemplo de FlowMode, quando nao sabe o que fazer a
    ↪ mensagem, enviar ao controlador.
# Tambem nao deve guardar no buffer do equipamento.
actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
    ↪ ofproto.OFPCML_NO_BUFFER)]
```

```
# Dois exemplos de FLOWMods.
# Caso destino esteja ligado ao mesmo switch/dpid, out_port =
    ↪ porta do destino, senao fazer flood

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD
```

2.3 Testes e Resultados

Para testar a comunicação entre o *controller* e o *OvSwitch*, o grupo começou por analisar uma captura do *Wireshark* antes dos fluxos de comunicação serem estabelecidos.

Note que foi aplicado o filtro:

openflow_v4 &&

!openflow_v4.type == OFPT_ECHO_REPLY &&

!openflow_v4.type == OFPT_ECHO_REQUEST

para apenas apresentar as mensagens relevantes.

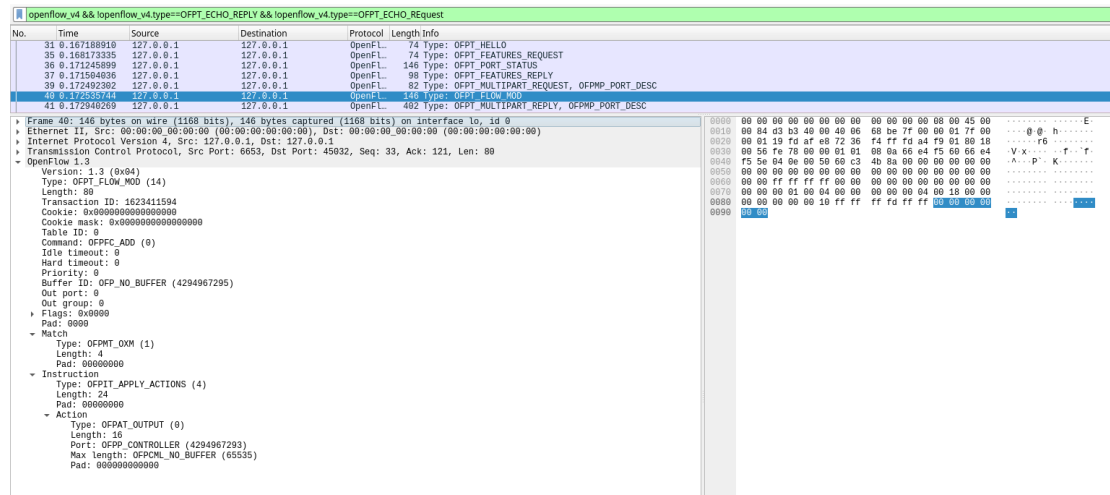


Figure 6: Captura do *Wireshark* antes de os fluxos serem estabelecidos.

A figura acima mostra a parte inicial de configuração da rede, que corresponde ao evento *EventOFPSwitchFeatures*, e instrui os *switches* a enviar todo o tráfego recebido para o *controller*. Esta regra tem prioridade 0, sendo equivalente a enviar pacotes para o *switch* quando não fazem *match* com nenhuma outra regra.

Posteriormente, o grupo fez uso do comando *pingall* e conseguiu visualizar a troca de pacotes *PACKET_IN* e *PACKET_OUT* e todos os *FLOW_MODS* estabelecidos para ocorrer a comunicação. Isto pode ser verificado pela figura abaixo.

openflow_v4 && !openflow_v4.type==OFPT_ECHO_REPLY && !openflow_v4.type==OFPT_ECHO_Request						
No.	Time	Source	Destination	Protocol	Length	Info
4	4.76917201	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
5	4.76993201	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
7	4.770772424	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
8	4.771677088	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
9	4.771724651	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
11	4.772441161	127.0.0.1	127.0.0.1	OpenFL	206	Type: OFPT_PACKET_IN
12	4.773074838	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
13	4.773098165	127.0.0.1	127.0.0.1	OpenFL	204	Type: OFPT_PACKET_OUT
15	4.776999962	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
16	4.777914544	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
17	4.778518469	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
18	4.779353082	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
19	4.779373406	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
21	4.779815087	127.0.0.1	127.0.0.1	OpenFL	206	Type: OFPT_PACKET_IN
22	4.780302704	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
23	4.780323188	127.0.0.1	127.0.0.1	OpenFL	204	Type: OFPT_PACKET_OUT
25	4.784126711	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
26	4.784836956	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
27	4.785610931	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
28	4.786359289	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
29	4.786384572	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
31	4.786882057	127.0.0.1	127.0.0.1	OpenFL	206	Type: OFPT_PACKET_IN
32	4.787493455	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
33	4.787514896	127.0.0.1	127.0.0.1	OpenFL	204	Type: OFPT_PACKET_OUT
35	4.793210807	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
36	4.793813883	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
37	4.794335254	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
38	4.794859860	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
39	4.794879324	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
41	4.795379882	127.0.0.1	127.0.0.1	OpenFL	206	Type: OFPT_PACKET_IN
42	4.796002943	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
43	4.796024804	127.0.0.1	127.0.0.1	OpenFL	204	Type: OFPT_PACKET_OUT
45	4.799510482	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
46	4.800356898	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
47	4.801808564	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
48	4.802058670	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
49	4.802092334	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
51	4.802707154	127.0.0.1	127.0.0.1	OpenFL	206	Type: OFPT_PACKET_IN
52	4.80390349	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
53	4.803421987	127.0.0.1	127.0.0.1	OpenFL	204	Type: OFPT_PACKET_OUT
55	4.811514589	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
56	4.812119281	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
57	4.812711333	127.0.0.1	127.0.0.1	OpenFL	150	Type: OFPT_PACKET_IN
58	4.813493634	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
59	4.813519894	127.0.0.1	127.0.0.1	OpenFL	148	Type: OFPT_PACKET_OUT
61	4.814138486	127.0.0.1	127.0.0.1	OpenFL	206	Type: OFPT_PACKET_IN
62	4.814937828	127.0.0.1	127.0.0.1	OpenFL	170	Type: OFPT_FLOW_MOD
63	4.814906533	127.0.0.1	127.0.0.1	OpenFL	204	Type: OFPT_PACKET_OUT
65	5.237234998	127.0.0.1	127.0.0.1	OpenFL	178	Type: OFPT_PACKET_IN
66	5.237690787	127.0.0.1	127.0.0.1	OpenFL	178	Type: OFPT_PACKET_IN

Figure 7: *Pingall*.

Na figura seguinte podemos ver com mais detalhe quais as mensagens trocadas entre os dispositivos (ARP Request, ARP Reply, FlowMod, Ping Request, etc).

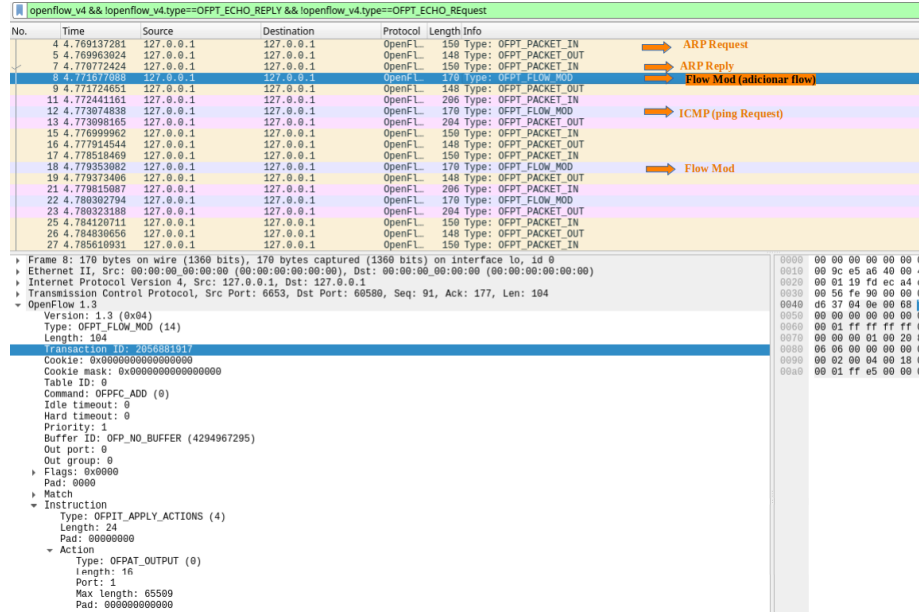


Figure 8: Captura do *Wireshark* depois de os fluxos serem estabelecidos.

Para visualizar os fluxos estabelecidos no *OvSwitch*, o grupo utilizou o comando **sudo ovs-ofctl dump-flows s1** e o resultado obtido foi o observado pela figura abaixo, na qual vemos todas as ligações entre o *switch* e as interfaces a que este está ligado. Qualquer repetição *switch-interface* corresponde a um *match*. Logo, o total de *flows* é 13, 1 para o controlador, e 3 *flows* para cada *host*.

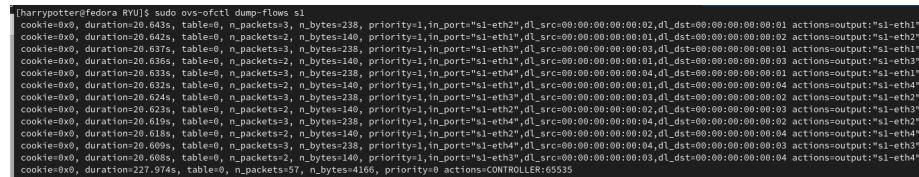


Figure 9: Visualização dos fluxos estabelecidos no *OvSwitch*.

3 Exercício 2

3.1 Estratégia adotada

Para descrever as várias informações no ficheiro de output, começamos por procurar obter duas informações:

1. Endereços MAC e IP de cada um dos elementos da topologia;
2. Ligações entre os vários nodos da topologia;

Para o primeiro ponto, utilizámos o evento *EventHostAdd* para descobrir os endereços *MAC* de todos os nodos. De seguida, é preciso que sejam trocadas mensagens com o protocolo *IPv4*, para que o evento *EventOFPPacketIn* seja ativado, e o controlador consiga associar *IPs* a *MACs*, através das informações das mensagens recebidas. Como os *switches* não tem *IPs*, podemos concluir que os nodos sem *IPs* são *switches*. Depois temos de associar os vários *MACs* a um mesmo *switch*, mas isso será descrito a seguir.

Quanto ao segundo conjunto de informações, quando um *Host* é adicionado à topologia, também armazenamos a que *switch* está associado, num dicionário cuja chave é o *dpid* do *switch*, e o valor o conjunto de portas e respetivo estado. Esse estado é inicializado a 1 (*up*) no início. Quando uma porta é modificada, identificável pelo evento *EventPortModify*, o estado da ligação é alterado.

Por fim, e de forma a associarmos *IPs* a *MACs*, nós verificamos cada pacote que chega ao controlador, e associamos os endereços de origem e de destino de nível 2 e 3.

Por fim, quando uma ligação é alterada, o ficheiro de saída é adaptado.

3.2 Testes e Resultados

Os testes que realizamos corresponderam à verificação do ficheiro de saída, se corresponde à topologia esperada. Para isso, começamos por iniciar a topologia, tendo o seguinte resultado, após efetuarmos o comando *pingall*. Só desta forma conseguimos saber os *IPs* dos *hosts*, e distingui-los dos *switches*.

```
hosts:
h01:
  ip: 10.0.0.1
  mac: 00:00:00:00:00:01
h02:
  ip: 10.0.0.2
  mac: 00:00:00:00:00:02
h03:
  ip: 10.0.0.3
  mac: 00:00:00:00:00:03
h04:
  ip: 10.0.0.4
  mac: 00:00:00:00:00:04
```

```
h05:
  ip: 10.0.0.5
  mac: 00:00:00:00:00:05
h06:
  ip: 10.0.0.6
  mac: 00:00:00:00:00:06
h07:
  ip: 10.0.0.7
  mac: 00:00:00:00:00:07
h08:
  ip: 10.0.0.8
  mac: 00:00:00:00:00:08
id1:
- 0a:2a:76:73:a5:8a
- 92:8a:ef:fe:4b:7c
id2:
- c2:dd:2d:b1:57:6b
id3:
- b6:2a:30:c3:b1:f6
switches:
id1:
  1:
    mac: 0a:2a:76:73:a5:8a
    status: 1
  2:
    mac: 92:8a:ef:fe:4b:7c
    status: 1
id2:
  1:
    mac: c2:dd:2d:b1:57:6b
    status: 1
  2:
    mac: 00:00:00:00:00:01
    status: 1
  3:
    mac: 00:00:00:00:00:02
    status: 1
  4:
    mac: 00:00:00:00:00:03
    status: 1
  5:
    mac: 00:00:00:00:00:04
    status: 1
id3:
  1:
    mac: b6:2a:30:c3:b1:f6
```

```

    status: 1
2:
    mac: 00:00:00:00:00:05
    status: 1
3:
    mac: 00:00:00:00:00:06
    status: 1
4:
    mac: 00:00:00:00:00:07
    status: 1
5:
    mac: 00:00:00:00:00:08
    status: 1

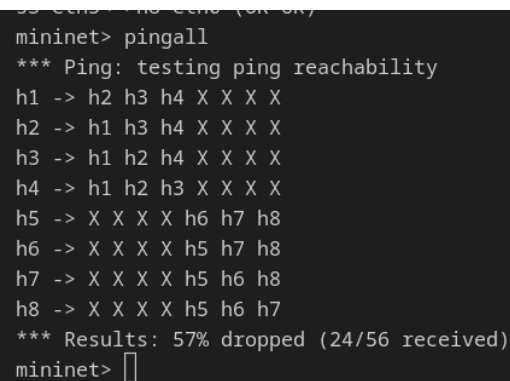
```

De seguida, desligamos a ligação entre dois *switches*, e executamos o comando *pingall*, verificando que não existe conexão entre os dois conjuntos de *hosts*. Para além disso, o estado da ligação no ficheiro foi alterado. Essa alteração encontra-se a seguir, sendo que o resto do ficheiro permaneceu igual:

```

(...)
switches:
  id1:
    1:
      mac: 0a:2a:76:73:a5:8a
      status: 0
    2:
      mac: 92:8a:ef:fe:4b:7c
      status: 1
(...)

```



```

mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4 X X X X
h2 -> h1 h3 h4 X X X X
h3 -> h1 h2 h4 X X X X
h4 -> h1 h2 h3 X X X X
h5 -> X X X X h6 h7 h8
h6 -> X X X X h5 h7 h8
h7 -> X X X X h5 h6 h8
h8 -> X X X X h5 h6 h7
*** Results: 57% dropped (24/56 received)
mininet>

```

Figure 10: Comando *pingall* quando se retira a ligação entre dois *switches*.

Alterando de novo a ligação, para que volte a ficar *up*, verificamos que o novo ficheiro corresponde ao estado inicial.

4 Conclusão

Com o realizar deste trabalho, o grupo considera que ganhou experiência prática com o protocolo *OpenFlow* e controladores SDN. Através dos dois exercícios apresentados, os alunos aprenderam a desenvolver aplicações que implementam um *switch* de autoaprendizagem de *Layer-2* e uma topologia de autoaprendizagem na rede. Ao utilizar as ferramentas de software necessárias, como *Mininet*, *Ryu* e *Wireshark*, os alunos conseguiram obter habilidades práticas em programação e configuração de rede.

5 Anexos

5.1 Código *simple_switch_13.py* comentado

```
def __init__(self, *args, **kwargs):
    (...)
    # Dicionário(s) device -> (mac_addr -> port)
    self.mac_to_port = {}

# Subscrever ao evento EventOFPSwitchFeatures, na fase de CONFIG
@set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
def switch_features_handler(self, ev):
    (...)
    # Match = qualquer pacote
    match = parser.OFPMatch()

    # Action = Output para Controlador
    actions = [parser.OFPActionOutput(
        ofproto.OFPP_CONTROLLER,
        ofproto.OFPCML_NO_BUFFER)]

    # Priority = 0
    self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions):
    (...)
    # Intruction = aplicar Action
    inst = [parser.OFPInstructionActions(
        ofproto.OFPIT_APPLY_ACTIONS,
        actions)]

    (...)
    # Contruir Flow Mod
    mod = parser.OFPFlowMod(
        datapath=datapath, priority=priority,
        match=match, instructions=inst)

    # Enviar Flow Mod para switch
    datapath.send_msg(mod)

# Subscrever ao evento EventOFPPacketIn, na fase "MAIN"
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    (...)
```

```

# "Ler" porta de entrada
in_port = msg.match['in_port']

# Reconstruir pacote apartir de msg.data
pkt = packet.Packet(msg.data)

# "Ler" encapsulamento Ethernet
eth = pkt.get_protocols(ethernet.ethernet)[0]

(...)

# "Ler" MAC address de destino
dst = eth.dst
# "Ler" MAC address da origem
src = eth.src

# Id do switch (int to string & pad 0 à esquerda)
dpid = format(datapath.id, "d").zfill(16)

# Caso dpid não exista, para não dar erro
self.mac_to_port.setdefault(dpid, {})

(...)

# learn a mac address to avoid FLOOD next time.
# Adicionar entrada dpid -> ( mac_addr_src -> in_port )
self.mac_to_port[dpid][src] = in_port

# Caso destino esteja ligado ao mesmo switch,
# out_port = porta do destino, senão fazer flood
if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD

# Action = Output para porta de saída
actions = [parser.OFPACTIONOutput(out_port)]

# install a flow to avoid packet_in next time
# Sabendo o in_port e out_port (not flood),
# adicionar flow ao switch
if out_port != ofproto.OFPP_FLOOD:
    # Match = mesma porta, origem e destino
    match = parser.OFPMATCH(
        in_port=in_port, eth_dst=dst,
        eth_src=src)

```

```

(...)

# Priority = 1
self.add_flow(datapath, 1, match, actions)

(...)

# Switch envia pacote original
data = msg.data
out = parser.OFPPacketOut(
    datapath=datapath, buffer_id=msg.buffer_id,
    in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)

```