Redes Definidas por Software

Relatório Lab 3

SDN controller

ONOS controller

Grupo 6
Miguel Carreiro, nº 82012

Exercício 4.1

Configuração da rede

Para a realização deste exercício, foi necessário implementar uma instância do controlador ONOS. Embora existam várias soluções para a implementação da mesma (Instalação direta, VM, container Docker), a solução escolhida foi a de criar um container em Docker, pois permite uma instalação e configuração simples.

A instalação e configuração do ONOS foi efetuada segundo o tutorial que se encontra na wiki do Project ONOS, com algumas alterações, nomeadamente, no comando de arranque do container, onde foi necessário adicionar ao container a exposição da porta correspondente ao OpenFlow (6653). Assim, a lista de comandos usados foi a seguinte:

- Obtenção da imagem do ONOS:

  ```
  $ docker pull onosproject/onos
  ```

- Criação e execução do container que irá correr o ONOS:

  ```
  $ docker run -t -d -p 8181:8181 -p 8101:8101 -p 5005:5005
  -p 830:830 -p 6653:6653 --name onos onosproject/onos
  ```

Após a execução destes comandos, o ONOS é iniciado, e as portas necessárias para o correto funcionamento do ONOS (8181, 8101, 5005, 830, 6653) ficam acessíveis a partir do host, sendo necessário para que se possa aceder à GUI, bem como para que o mininet possa comunicar com o controlador ONOS.

Uma vez que a instalação do ONOS foi efetuada de uma forma diferente da descrita no enunciado, a captura de pacotes da interface localhost da máquina virtual foi substituída pela captura de outras interfaces, nomeadamente:

- A interface docker0, que efetua a bridge entre o host e o container onde corre o ONOS, e onde se podem capturar todos os pacotes que são transferidos entre o controlador ONOS e o Mininet;
- As interfaces dos switches do Mininet, onde é possível capturar os pacotes que entram ou saem dessa mesma interface, o que permite observar o tráfego em todos os links que ligam os switches, bem como os links que ligam os switches aos hosts (apesar de não ser possível capturar o tráfego das interfaces dos hosts, como todos eles se encontram ligados a um switch, basta capturar a interface do switch onde o host se liga).

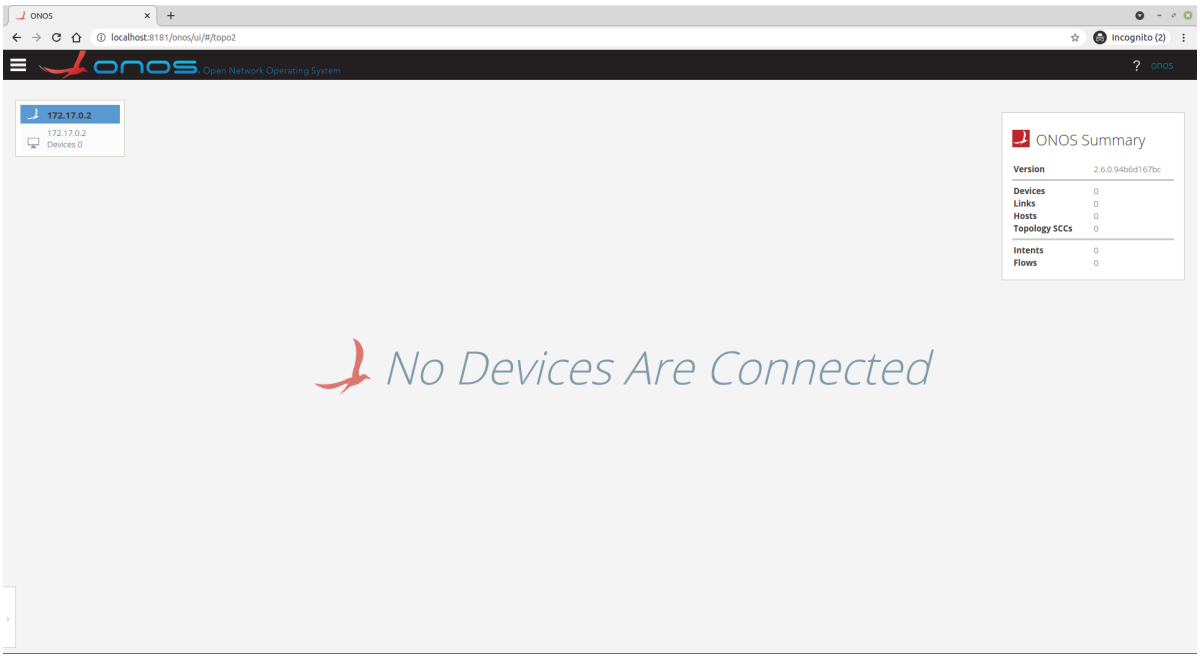
Execução dos comandos

Inicialização do ONOS

O ONOS foi inicializado usando os comandos referidos anteriormente (que não correspondem ao do ponto 1, pois a forma de instalação é diferente), e a GUI ficou acessível através do endereço http://localhost:8181/onos/ui/login.html.

4 - Verificar dispositivos conectados no ONOS

Após arrancar o Mininet usando o comando do ponto 3, e efetuar o login na GUI do ONOS, foi possível observar que não foi detetado qualquer dispositivo conectado, como se pode ver na figura seguinte:



Ao observar a captura da interface docker0, podemos observar que existe troca de pacotes TCP nas portas 6653 (correspondente ao protocolo OpenFlow, com origem e/ou destino no mininet) e 8181 (correspondente à troca de pacotes entre o browser com a GUI aberta e o ONOS):

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.1	172.17.0.2	TCP	74	38674 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699265542 TSecr=0 WS=512
2	0.000074672	172.17.0.2	172.17.0.1	TCP	54	6653 → 38674 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
3	0.000331752	172.17.0.1	172.17.0.2	TCP	74	38676 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699265542 TSecr=0 WS=512
4	0.000372801	172.17.0.2	172.17.0.1	TCP	54	6653 → 38676 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
5	0.063403632	172.17.0.2	172.17.0.1	TCP	68	8181 → 39830 [PSH, ACK] Seq=1 Ack=1 Win=84 Len=2 TSval=3753165563 TSecr=2699255606
6	0.063475392	172.17.0.1	172.17.0.2	TCP	66	39830 → 8181 [ACK] Seq=1 Ack=3 Win=83 Len=0 TSval=2699265606 TSecr=3753165563
7	0.063863725	172.17.0.1	172.17.0.2	TCP	72	39830 → 8181 [PSH, ACK] Seq=1 Ack=3 Win=83 Len=6 TSval=2699265606 TSecr=3753165563
8	0.063899022	172.17.0.2	172.17.0.1	TCP	66	8181 → 39830 [ACK] Seq=3 Ack=7 Win=84 Len=0 TSval=3753165563 TSecr=2699265606
9	1.000575100	172.17.0.1	172.17.0.2	TCP	74	38682 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699266543 TSecr=0 WS=512
10	1.000650203	172.17.0.2	172.17.0.1	TCP	54	6653 → 38682 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
11	1.001392239	172.17.0.1	172.17.0.2	TCP	74	38684 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699266544 TSecr=0 WS=512
12	1.001437654	172.17.0.2	172.17.0.1	TCP	54	6653 → 38684 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
13	1.999878349	172.17.0.1	172.17.0.2	TCP	74	38690 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699267542 TSecr=0 WS=512
14	1.999889906	172.17.0.1	172.17.0.2	TCP	74	38692 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699267542 TSecr=0 WS=512
15	1.999943189	172.17.0.2	172.17.0.1	TCP	54	6653 → 38692 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
16	1.999943299	172.17.0.2	172.17.0.1	TCP	54	6653 → 38690 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
17	2.999985487	172.17.0.1	172.17.0.2	TCP	74	38698 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699268542 TSecr=0 WS=512
18	3.000055479	172.17.0.2	172.17.0.1	TCP	54	6653 → 38698 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
19	3.000413918	172.17.0.1	172.17.0.2	TCP	74	38700 → 6653 [SYN] Seq=0 Win=42340 Len=0 MSS=1460 SACK_PERM=1 TSval=2699268543 TSecr=0 WS=512
20	3.000449109	172.17.0.2	172.17.0.1	TCP	54	6653 → 38700 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Observando com maior detalhe os pacotes trocados de e para a porta 6653 do container, podemos verificar que o container, ao receber um pacote TCP, responde enviando um pacote TCP ACK com a flag Reset igual a 1, o que significa que o container (ou neste caso, o ONOS) não esperava nem pretende receber este tipo de pacote. Uma vez que a porta 6653 serve para a troca de pacotes OpenFlow entre os switches e o controlador, o facto dos switches receberem pacotes com a flag Reset ativa significa que algo não está a funcionar corretamente no controlador. Isto deve-se ao facto da aplicação que permite ao ONOS interpretar os pacotes OpenFlow não se encontrar ativa.

6 - Correr Mininet após ativar app "OpenFlow Provider Suite"

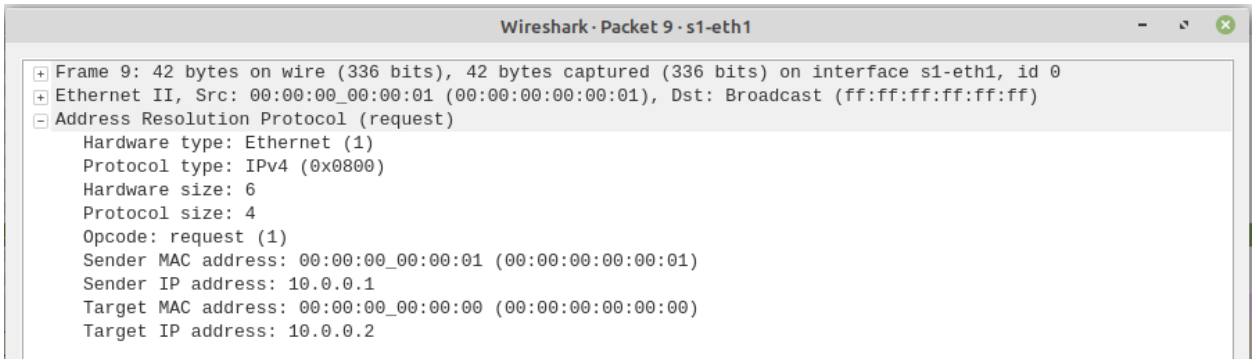
Após ativar a aplicação "OpenFlow Provider Suite", e correr novamente o Mininet, é possível confirmar que, tal como é referido no enunciado, os switches já aparecem na GUI:

7 - Verificar conectividade

Ao executar o comando `mininet> h1 ping -c5 h2`, os pings não são concluídos com sucesso, pois ocorre o erro "Destination Host Unreachable":

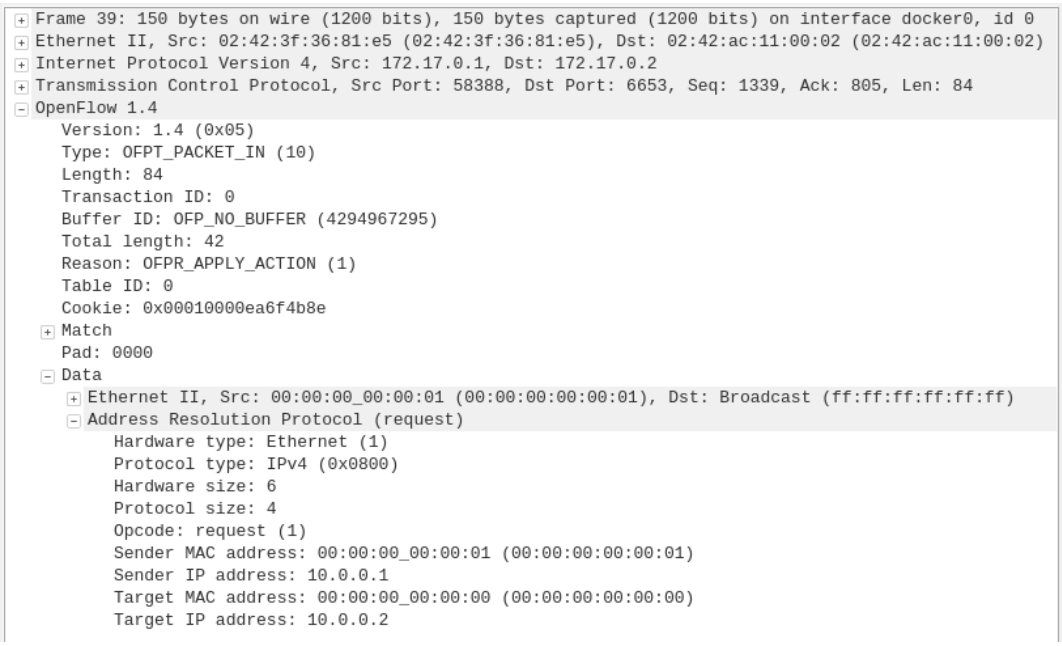
```
mininet> h1 ping -c5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Destination Host Unreachable
From 10.0.0.1 icmp_seq=2 Destination Host Unreachable
From 10.0.0.1 icmp_seq=3 Destination Host Unreachable
From 10.0.0.1 icmp_seq=4 Destination Host Unreachable
From 10.0.0.1 icmp_seq=5 Destination Host Unreachable

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, +5 errors, 100% packet loss, time 4093ms
```

Ao observar as capturas do link entre os switches s1 e s2, é possível observar o envio de pacotes ARP request desde o host 1, e como destino o endereço MAC de broadcast:

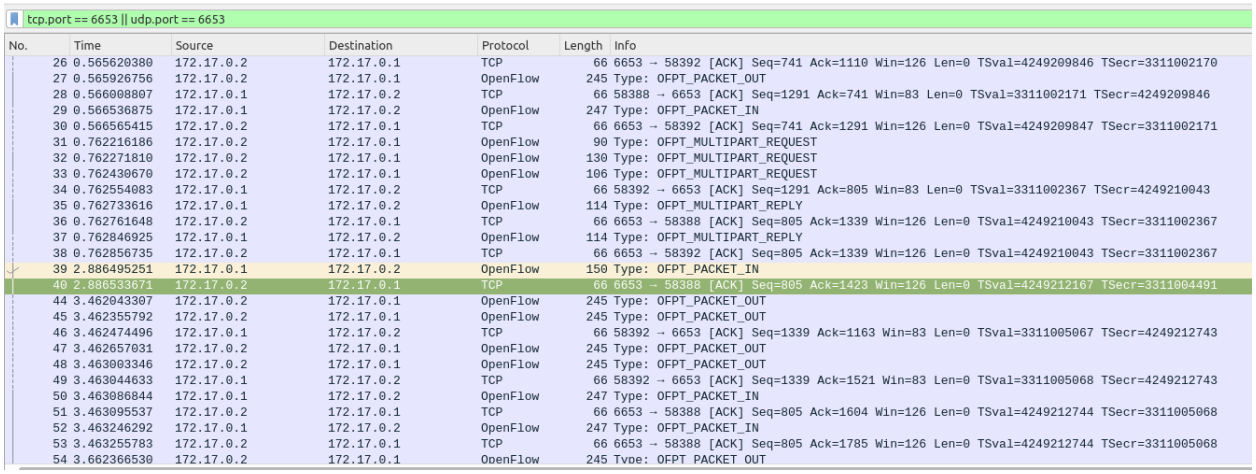


Analisando os pacotes trocados no link, constata-se que não houve nenhuma resposta aos pacotes ARP request enviados, pelo que o host, não conhecendo o endereço MAC de destino (h2), não poderá enviar os pacotes ICMP request (ping).

Após a análise dos pacotes enviados dentro da topologia, torna-se necessário analisar os pacotes trocados entre o Mininet e o ONOS, sendo que, nesta situação, a captura foi filtrada de forma a mostrar apenas os pacotes que tenham origem ou destino na porta 6653, correspondente aos pacotes OpenFlow que são trocados entre o Mininet e o ONOS. Assim, ao ser efetuado o ping entre o h1 e o h2, é possível observar o envio de um pacote OpenFlow do tipo OFPT_PACKET_IN, que inclui o ARP request anteriormente enviado pelo h1:



Observando a lista de pacotes enviados após este pacote, podemos ver que o ONOS envia um pacote TCP com a flag ACK, o que significa que recebeu corretamente o pacote:



tcp.port == 6653 || udp.port == 6653

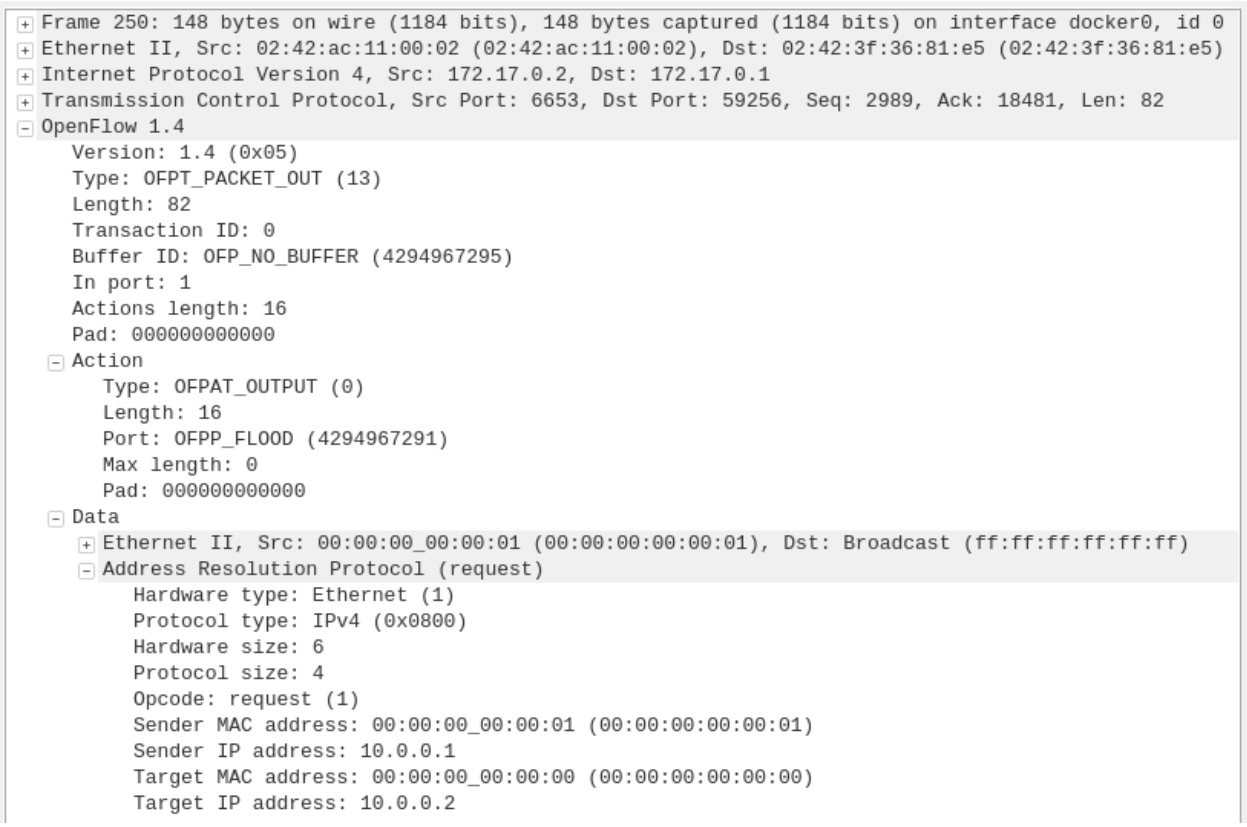
No.	Time	Source	Destination	Protocol	Length	Info
26	0.565620380	172.17.0.2	172.17.0.1	TCP	66	6653 → 58392 [ACK] Seq=741 Ack=1110 Win=126 Len=0 TSval=4249209846 TSecr=3311002170
27	0.565926756	172.17.0.2	172.17.0.1	OpenFlow	245	Type: OFPT_PACKET_OUT
28	0.566008807	172.17.0.1	172.17.0.2	TCP	66	58388 → 6653 [ACK] Seq=1291 Ack=741 Win=83 Len=0 TSval=3311002171 TSecr=4249209846
29	0.566536875	172.17.0.1	172.17.0.2	OpenFlow	247	Type: OFPT_PACKET_IN
30	0.566565415	172.17.0.2	172.17.0.1	TCP	66	6653 → 58392 [ACK] Seq=741 Ack=1291 Win=126 Len=0 TSval=4249209847 TSecr=3311002171
31	0.762216186	172.17.0.2	172.17.0.1	OpenFlow	90	Type: OFPT_MULTIPART_REQUEST
32	0.762271810	172.17.0.2	172.17.0.1	OpenFlow	130	Type: OFPT_MULTIPART_REQUEST
33	0.762430670	172.17.0.2	172.17.0.1	OpenFlow	106	Type: OFPT_MULTIPART_REQUEST
34	0.762554083	172.17.0.1	172.17.0.2	TCP	66	58392 → 6653 [ACK] Seq=1291 Ack=805 Win=83 Len=0 TSval=3311002367 TSecr=4249210043
35	0.762733616	172.17.0.1	172.17.0.2	OpenFlow	114	Type: OFPT_MULTIPART_REPLY
36	0.762761648	172.17.0.2	172.17.0.1	TCP	66	6653 → 58388 [ACK] Seq=805 Ack=1339 Win=126 Len=0 TSval=4249210043 TSecr=3311002367
37	0.762846925	172.17.0.1	172.17.0.2	OpenFlow	114	Type: OFPT_MULTIPART_REPLY
38	0.762856735	172.17.0.2	172.17.0.1	TCP	66	6653 → 58392 [ACK] Seq=805 Ack=1339 Win=126 Len=0 TSval=4249210043 TSecr=3311002367
39	2.886495251	172.17.0.1	172.17.0.2	OpenFlow	150	Type: OFPT_PACKET_IN
40	2.886533671	172.17.0.2	172.17.0.1	TCP	66	6653 → 58388 [ACK] Seq=805 Ack=1423 Win=126 Len=0 TSval=4249212167 TSecr=3311004491
44	3.462043307	172.17.0.2	172.17.0.1	OpenFlow	245	Type: OFPT_PACKET_OUT
45	3.462355792	172.17.0.2	172.17.0.1	OpenFlow	245	Type: OFPT_PACKET_OUT
46	3.462474496	172.17.0.1	172.17.0.2	TCP	66	58392 → 6653 [ACK] Seq=1339 Ack=1163 Win=83 Len=0 TSval=3311005067 TSecr=4249212743
47	3.462657031	172.17.0.2	172.17.0.1	OpenFlow	245	Type: OFPT_PACKET_OUT
48	3.463003346	172.17.0.2	172.17.0.1	OpenFlow	245	Type: OFPT_PACKET_OUT
49	3.463044633	172.17.0.1	172.17.0.2	TCP	66	58392 → 6653 [ACK] Seq=1339 Ack=1521 Win=83 Len=0 TSval=3311005068 TSecr=4249212743
50	3.463086844	172.17.0.1	172.17.0.2	OpenFlow	247	Type: OFPT_PACKET_IN
51	3.463095537	172.17.0.2	172.17.0.1	TCP	66	6653 → 58388 [ACK] Seq=805 Ack=1604 Win=126 Len=0 TSval=4249212744 TSecr=3311005068
52	3.463246292	172.17.0.1	172.17.0.2	OpenFlow	247	Type: OFPT_PACKET_IN
53	3.463255783	172.17.0.2	172.17.0.1	TCP	66	6653 → 58388 [ACK] Seq=805 Ack=1785 Win=126 Len=0 TSval=4249212744 TSecr=3311005068
54	3.662366530	172.17.0.2	172.17.0.1	OpenFlow	245	Type: OFPT_PACKET_OUT

Nota: Ainda que se possam observar alguns pacotes OFPT_PACKET_OUT, os mesmos são enviados periodicamente, mesmo antes do ping, pelo que não proporcionam dados adicionais que permitam um correto encaminhamento dos pacotes por parte dos switches.

Retornando ao pacote ARP, ainda que o ONOS o tenha recebido corretamente, o controlador não tem a app “Reactive Forwarding” ativa, pelo que não envia para o Mininet a informação acerca do encaminhamento do pacote, pelo que o pacote ARP não chega ao destino, e, consequentemente, o host 1 não consegue obter o endereço MAC de destino, e não poderá enviar o pacote ICMP request.

8 - Ativar app “Reactive Forwarding” e verificar conectividade

Ao ativar a app “Reactive Forwarding”, o ONOS passa a responder aos pacotes OFPT_PACKET_IN enviados pelo Mininet, começando pelo pacote ARP request, cujo ONOS indicou a “porta” flood (que corresponde ao envio do pacote para todas as portas menos a que recebeu o mesmo):



Após o envio desta instrução para o switch 1 no Mininet, o mesmo enviou o pacote ARP request por broadcast, que foi recebido pelo switch 2, que efetuou o mesmo processo do switch 1 para saber para onde encaminhar o pacote (envio do pacote para o ONOS, tendo recebido a instrução "flood"). Após este processo, o pacote ARP request chegou ao host 2, que respondeu com o envio de um pacote ARP reply, incluindo o seu endereço MAC, e tendo como destino o host 1. O switch 2, ao receber este pacote, envia-o para o ONOS, no entanto, desta vez o endereço de destino é unicast, ao contrário do pacote ARP request (endereço broadcast), pelo que, o ONOS (mais especificamente a app "Reactive Forwarding", que usa o "ONOS path service" para calcular a rota mais curta), não tendo nenhum flow manual configurado, retorna a porta para onde o switch deve reencaminhar o pacote tendo em conta a rota mais curta para o destino. Este processo repete-se quando o pacote chega ao switch 1, e, consequentemente, o pacote ARP reply chega ao host 1, que agora tem o endereço MAC para o qual deve enviar os pacotes ICMP.

Como se pode ver na captura do Mininet, o host 1 consegue efetuar o ping para o host 2 sem qualquer problema:

```
mininet> h1 ping -c5 h2
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=13.2 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.779 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.090 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.080 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.135 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4062ms
rtt min/avg/max/mdev = 0.080/2.876/13.297/5.217 ms
```

Observando as capturas da interface s1-eth2 (correspondente ao link s1-s2), é possível confirmar que existe conectividade entre os switches e, consequentemente, entre os hosts:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	MA/00:00:00:00:00:01 PC/32 120
2	0.000203558	02:eb:9f:67:c9:42	Broadcast	LLDP	139	MA/00:00:00:00:00:01 PC/32 120
3	0.101165357	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	MA/00:00:00:00:00:02 PC/32 120
4	0.101336129	02:eb:9f:67:c9:42	Broadcast	LLDP	139	MA/00:00:00:00:00:02 PC/32 120
5	1.695341040	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x3ef7, seq=1/256, ttl=64 (reply in 6)
6	1.698532152	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3ef7, seq=1/256, ttl=64 (request in 5)
7	2.694622926	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x3ef7, seq=2/512, ttl=64 (reply in 8)
8	2.695109447	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3ef7, seq=2/512, ttl=64 (request in 7)
9	3.100662055	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	MA/00:00:00:00:00:01 PC/32 120
10	3.100843346	02:eb:9f:67:c9:42	Broadcast	LLDP	139	MA/00:00:00:00:00:01 PC/32 120
11	3.200137203	02:eb:9f:67:c9:42	LLDP_Multicast	LLDP	139	MA/00:00:00:00:00:02 PC/32 120
12	3.200240903	02:eb:9f:67:c9:42	Broadcast	LLDP	139	MA/00:00:00:00:00:02 PC/32 120
13	3.695500062	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x3ef7, seq=3/768, ttl=64 (reply in 14)
14	3.695535024	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3ef7, seq=3/768, ttl=64 (request in 13)
15	4.708566495	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x3ef7, seq=4/1024, ttl=64 (reply in 16)
16	4.708598352	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3ef7, seq=4/1024, ttl=64 (request in 15)
17	5.728605147	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) request id=0x3ef7, seq=5/1280, ttl=64 (reply in 18)
18	5.728649931	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) reply id=0x3ef7, seq=5/1280, ttl=64 (request in 17)

Exercício 4.2

Configuração da rede

A instalação e o arranque do controlador ONOS foi feita da mesma forma do que no exercício 4.1, ou seja, através de um container Docker. A configuração da topologia no Mininet, bem como a configuração da largura de banda máxima e delay dos links, foi feita usando o mesmo ficheiro de configuração usado no laboratório anterior, dado que os parâmetros são semelhantes. O comando usado para arrancar o Mininet foi o seguinte:

```
$ sudo mn --custom 4-2_Topology.py --topo mytopo --mac
                    --controller=remote
```

O ficheiro usado para a configuração da topologia (4-2_Topology.py) encontra-se no zip submetido no Fénix.

A configuração dos flows no ONOS foi feita usando a sua REST API, sendo que foi usada a ferramenta cURL para efetuar os pedidos à API. O comando usado para adicionar os flows foi o seguinte:

```
$ curl -X POST --header 'Content-Type: application/json' --header
            'Accept: application/json' -d @flows.json
      'http://localhost:8181/onos/v1/flows' --user onos:rocks
```

O ficheiro flows.json, que inclui todos os flows adicionados, encontra-se no zip submetido no Fénix. Os flows adicionados correspondem a ter todos os pacotes com origem ou destino no h3 a passar pelo link S1-S3, todos os pacotes com origem ou destino no h4 a passar pelo link S2-S3, e, nos restantes casos, é usada a rota mais curta.

Testes de conectividade

Para verificar o correto funcionamento dos flows, foi usada a ferramenta iperf, medindo a largura de banda entre vários hosts. Uma vez que o link S1-S3 tem uma velocidade limitada a 100 Mb/s, ao contrário dos restantes links limitados a 1000 Mb/s, podemos aferir qual o trajeto que os pacotes efetuam entre os hosts.

Iniciando pela ligação entre os hosts 1 e 3, a mesma deverá ter uma velocidade máxima de 100 Mb/s (ou aproximado), uma vez que os pacotes devem passar pelo link S1-S3, com limitação de velocidade a 100 Mb/s. O resultado obtido foi o seguinte:

```
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: ['94.5 Mbits/sec', '110 Mbits/sec']
```

Confirma-se assim que a rota entre os hosts 1 e 3 funciona corretamente.

A rota que os pacotes que transitam entre os hosts 1 e 4 deve passar pelo S2, pelo que a velocidade máxima se deve aproximar dos 1000 Mb/s. Ao executar o iperf entre o h1 e o h4, o resultado foi o seguinte:

```
mininet> iperf h1 h4
*** Iperf: testing TCP bandwidth between h1 and h4
*** Results: ['949 Mbits/sec', '954 Mbits/sec']
```

Podemos observar que o resultado é bastante diferente do anterior (rota h1-h3), uma vez que o trajeto contempla apenas links com uma velocidade máxima de 1000 Mb/s, o que, embora seja um trajeto maior e com mais "saltos" (hops), tem menos restrições a nível de throughput e delay.

Por último, seguem-se os testes de conectividade entre o host 2 e os restantes:

```
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['948 Mbits/sec', '953 Mbits/sec']
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: ['94.6 Mbits/sec', '111 Mbits/sec']
mininet> iperf h2 h4
*** Iperf: testing TCP bandwidth between h2 and h4
*** Results: ['948 Mbits/sec', '954 Mbits/sec']
```

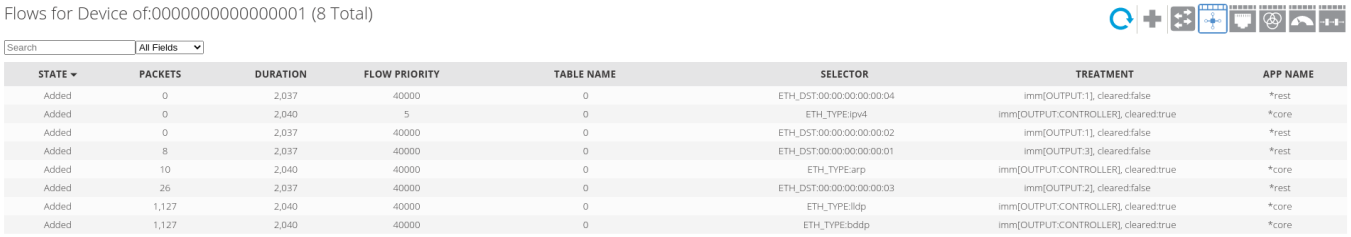
Como podemos observar, o host 2 tem conectividade com os restantes hosts, e, no caso da ligação com o host 3, é possível verificar que a mesma passa pelo link S1-S3, uma vez que a velocidade máxima corresponde aproximadamente à velocidade máxima desse link.

Eliminação de link com os flows configurados

Para verificar o que acontece neste tipo de situações, foram efetuados vários pings entre os hosts, sendo que existe uma parte das rotas (geradas através dos flows adicionados via REST API) que passa por um link que foi eliminado. Neste exemplo, o link eliminado foi o S1-S3, e os pings efetuados foram os seguintes:

```
mininet> h1 ping -c5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4076ms
```

Podemos observar que não foi possível efetuar o ping entre o h1 e o h3, por timeout. A tabela de flows do S1 logo após as tentativas de ping é a seguinte:



Flows for Device of:0000000000000001 (8 Total)

Search All Fields

STATE	PACKETS	DURATION	FLOW PRIORITY	TABLE NAME	SELECTOR	TREATMENT	APP NAME
Added	0	2,037	40000	0	ETH_DST:00:00:00:00:00:04	imm[OUTPUT:1], cleared:false	*rest
Added	0	2,040	5	0	ETH_TYPE:ipv4	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	0	2,037	40000	0	ETH_DST:00:00:00:00:00:02	imm[OUTPUT:1], cleared:false	*rest
Added	8	2,037	40000	0	ETH_DST:00:00:00:00:00:01	imm[OUTPUT:3], cleared:false	*rest
Added	10	2,040	40000	0	ETH_TYPE:arp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	26	2,037	40000	0	ETH_DST:00:00:00:00:00:03	imm[OUTPUT:2], cleared:false	*rest
Added	1,127	2,040	40000	0	ETH_TYPE:lldp	imm[OUTPUT:CONTROLLER], cleared:true	*core
Added	1,127	2,040	40000	0	ETH_TYPE:bddp	imm[OUTPUT:CONTROLLER], cleared:true	*core

Constata-se que, para além dos flows já criados anteriormente, não foi criado nenhum flow adicional, o que significa que, quando o h1 enviou o pacote ICMP request, o S1 tentou encaminhar o pacote para o link S1-S3, que se encontra desativado, implicando assim que o pacote não chegue ao destino (h3) e que, consequentemente, não seja possível ao h1 receber a resposta (pacote ICMP reply).

O resultado dos pings entre o h4 e o h1 são os seguintes:

```
mininet> h4 ping -c5 h1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.096 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.128 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.143 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.065 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.146 ms

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4078ms
rtt min/avg/max/mdev = 0.065/0.115/0.146/0.033 ms
```

Podemos observar que os pings são executados corretamente, pois os links pertencentes à rota definida pelos flows criados anteriormente não foram afetados.

Ao efetuar os pings no h2, e tendo como destino os hosts 3 e 4, os resultados são bastante diferentes:

```
mininet> h2 ping -c5 h3
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.

--- 10.0.0.3 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4078ms

mininet> h2 ping -c5 h4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=5.90 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.109 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=0.088 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=0.104 ms
64 bytes from 10.0.0.4: icmp_seq=5 ttl=64 time=0.077 ms

--- 10.0.0.4 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4079ms
rtt min/avg/max/mdev = 0.077/1.256/5.902/2.323 ms
```

Isto acontece pois a rota entre o h2 e o h3 (h2 <-> S2 <-> S1 <-> S3 <-> h3) é diferente da rota entre o h2 e o h4 (h2 <-> S2 <-> S3 <-> h4) [a vermelho - link desativado], e após a desativação do link não é criado nem alterado nenhum flow que possa corrigir as rotas entre os hosts.