Redes Definidas por Software

Relatório Lab 2

Introduction to Mininet and OpenFlow

Grupo 6
Miguel Carreiro, nº 82012

Ambiente de teste

Para a execução dos exercícios solicitados, foi usado o Mininet (v2.3.0) numa distribuição Linux. Para a captura dos pacotes foi usado o Wireshark.

O código e scripts utilizados encontram-se no zip submetido no Fénix.

Exercício 4.1

Configuração da rede

Para a realização deste exercício, não foi necessária qualquer configuração da topologia, uma vez que a mesma é criada aquando a execução do primeiro comando (uma análise mais detalhada do mesmo será feita mais à frente).

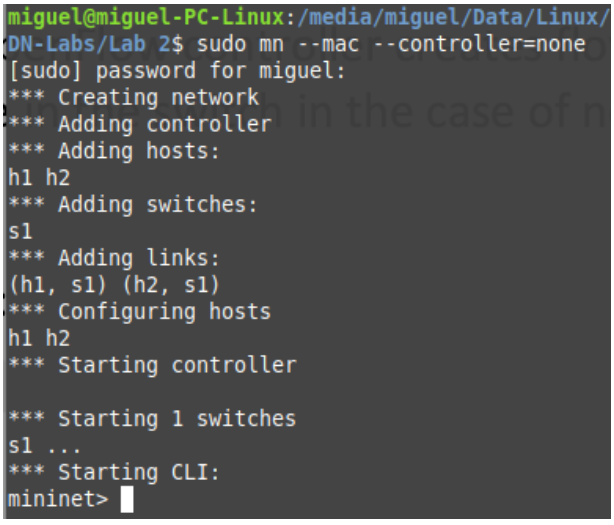
Execução dos comandos

Comando 1

```
$ sudo mn --mac --controller=none
```

Este comando inicia o Mininet, e, uma vez que não foi especificada nenhuma topologia, é utilizada uma topologia padrão (minimal), cuja representação se encontra na figura 1 do enunciado.

Ao correr o comando, podemos observar que, à medida que os componentes da topologia são criados, o terminal exibe a identificação dos mesmos, de forma a confirmar que o arranque é feito de forma correta. Nesta informação de arranque, podemos observar o seguinte:



Os hosts, switches e links são criados de acordo com a topologia, no entanto, não foi criado nenhum controlador. Tal aconteceu, uma vez que foi adicionada a flag `--controller=none`, que faz com que o mininet seja executado sem qualquer controlador.

A flag `--mac` será explicada na subsecção do comando 6.

Comando 2

```
mininet> h1 ping -c5 h2
```

Ao efetuar os pings, é possível constatar que, tal como refere o enunciado, não é possível concretizar o ping, uma vez que é devolvido 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 4076ms
```

Uma vez que não se encontra nenhum flow configurado, e não existe nenhum controlador, não é possível enviar e receber pacotes entre os hosts.

Comando 3

```
$ sudo ovs-ofctl dump-flows s1
```

O comando ovs-ofctl permite administrar, bem como obter o estado atual dos switches OpenFlow (configurações, funcionalidades e entradas de tabela). O comando ovs-ofctl comunica com os switches de várias formas, nomeadamente através de SSL, TCP, socket, bridge (através de um ficheiro de configuração), ou através da busca da bridge associada ao datapath.

Ao executar este comando, é possível constatar que não existe qualquer flow configurado no switch s1:

```
mininet> sh sudo ovs-ofctl dump-flows s1
```

(Nota: após a execução deste comando não é retornado qualquer texto e/ou conteúdo)

Existindo um controlador, os pacotes recebidos seriam encaminhados para o mesmo, mas, uma vez que esta topologia de teste não possui controlador, os

pacotes são rejeitados, uma vez que o switch não tem informações suficientes para determinar para onde encaminhar o pacote.

Comando 4

```
$ sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2
$ sudo ovs-ofctl add-flow s1 in_port=2,actions=output:1
```

Estes comandos têm como finalidade adicionar novos flows aos switches, o que permite redirecionar corretamente os pacotes.

Neste caso, o primeiro comando faz com que todos os pacotes que são recebidos através da interface eth1 (ou seja, vindos do h1) sejam encaminhados para a interface eth2 (ou seja, terão como destino o h2). Ainda assim, não é suficiente para garantir a conectividade entre os hosts, pois o pacote de resposta não será recebido, pois falta definir o flow correspondente à rota inversa (h2 -> h1), e é aqui que entra o segundo comando, que define o encaminhamento dos pacotes recebidos através da interface eth2 para a interface eth1.

Após a adição destes dois flows, a conectividade entre o h1 e h2 deverá funcionar, no entanto, a confirmação será feita no ponto seguinte.

5 - Testes de conectividade

Para testar a conectividade entre o h1 e o h2, foram efetuados pings entre os dois hosts (h1->h2 e h2->h1). Os resultados foram os seguintes:

```
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=0.298 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.111 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.132 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.100 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.102 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4094ms
rtt min/avg/max/mdev = 0.100/0.148/0.298/0.076 ms
mininet> h2 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=0.101 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.111 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.090 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.046 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.049 ms

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4087ms
rtt min/avg/max/mdev = 0.046/0.079/0.111/0.028 ms
```

Podemos observar que existe conectividade entre os hosts h1 e h2 (nas duas direções), confirmando assim que os flows adicionados permitem um correto encaminhamento dos pacotes entre o h1 e o h2.

Comando 6

```
$ sudo mn --mac
```

Este comando é praticamente igual ao comando 1, no entanto, não inclui a opção `--controller`, o que significa que, neste caso, o mininet irá arrancar com a topologia "minimal" completa (2 hosts, 1 switch e 1 controlador). A opção --mac faz com que os endereços MAC dos hosts sejam gerados de forma a ter uma leitura fácil e simplificada.

Ao executar o comando, o mininet cria e configura todos os elementos da rede (desta vez incluindo o controlador, como referido anteriormente). Os endereços MAC dos hosts possuem o formato 00:00:00:00:00:0n (n- número do host).

Comando 7

Abrir o Wireshark, e iniciar captura na interface loopback com o filtro openflow_v1

Ao iniciar a captura, podemos constatar que existe troca de pacotes entre o switch e o controlador, nomeadamente pacotes do tipo OFPT_ECHO_REQUEST e OFPT_ECHO_REPLY, que têm como finalidade a verificação constante da ligação entre o switch e o controlador, e pacotes do tipo OFPT_PACKET_IN, que contém pacotes IPv6 do tipo ICMPv6 Router Solicitation, enviados periodicamente pelos hosts, e que, uma vez que não existem flows configurados no switch, são reencaminhados para o controlador, que retorna um pacote do tipo OFPT_PACKET_OUT, e que o switch reencaminha para o destino (o host vizinho) a parte correspondente ao pacote ICMPv6.

Comando 8

```
mininet> h1 ping -c5 h2
```

Ao efetuar o primeiro ping, o h1 possui o endereço IP de destino, mas não possui o endereço MAC correspondente ao endereço IP, pelo que tem de enviar um pacote ARP request, de forma a que lhe seja retornado o endereço MAC para o qual terá de enviar o pacote Echo Request. O switch, uma vez que não possui flow entries para encaminhar o ARP request, encaminha o mesmo para o controlador, mas encapsulado num pacote do tipo OFPT_PACKET_IN, cuja captura é a seguinte:

```
⊞ Frame 213: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface lo, id 0
⊞ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
⊞ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
⊞ Transmission Control Protocol, Src Port: 57310, Dst Port: 6653, Seq: 1753, Ack: 1349, Len: 60
⊟ OpenFlow 1.0
    .000 0001 = Version: 1.0 (0x01)
    Type: OFPT_PACKET_IN (10)
    Length: 60
    Transaction ID: 0
    Buffer Id: 0xffffffff
    Total length: 42
    In port: 1
    Reason: No matching flow (table-miss flow entry) (0)
    Pad: 00
  ⊞ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊟ Address Resolution Protocol (request)
      Hardware type: Ethernet (1)
      Protocol type: IPv4 (0x0800)
      Hardware size: 6
      Protocol size: 4
      Opcode: request (1)
      Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
      Sender IP address: 10.0.0.1
      Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
      Target IP address: 10.0.0.2
```

Podemos observar que o pacote inclui o ARP request, bem como a indicação de que não existe nenhuma flow entry para este endereço MAC de destino (broadcast - ff.ff.ff.ff.ff.ff). O controlador, que possui os dados de encaminhamento, responde com um pacote OFPT_PACKET_OUT, com a instrução dada ao switch acerca do que fazer com o pacote:

```
⊞ Frame 214: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on interface lo, id 0
⊞ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
⊞ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
⊞ Transmission Control Protocol, Src Port: 6653, Dst Port: 57310, Seq: 1349, Ack: 1813, Len: 66
⊟ OpenFlow 1.0
    .000 0001 = Version: 1.0 (0x01)
    Type: OFPT_PACKET_OUT (13)
    Length: 66
    Transaction ID: 0
    Buffer Id: 0xffffffff
    In port: 1
    Actions length: 8
    Actions type: Output to switch port (0)
    Action length: 8
    Output port: 65531
    Max length: 0
  ⊞ Ethernet II, Src: 00:00:00_00:00:01 (00:00:00:00:00:01), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ⊟ Address Resolution Protocol (request)
      Hardware type: Ethernet (1)
      Protocol type: IPv4 (0x0800)
      Hardware size: 6
      Protocol size: 4
      Opcode: request (1)
      Sender MAC address: 00:00:00_00:00:01 (00:00:00:00:00:01)
      Sender IP address: 10.0.0.1
      Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
      Target IP address: 10.0.0.2
```

Como podemos observar, o controlador informa o switch de que deve reencaminhar o pacote para a porta 65531, que corresponde ao equivalente da opção “flood” do OpenFlow, que envia o pacote para todas as portas menos a que o recebeu. O h2, ao receber o ARP request, responde com o envio de um pacote ARP

reply, incluindo o seu endereço MAC, após o qual se efetua o mesmo procedimento efetuado aquando o envio do pacote ARP request, mas com a diferença que neste caso o endereço MAC de destino é o do h1, em vez do broadcast. Isto faz com que, o controlador indique a porta 1 (correspondente ao link com o h1), em vez de fazer o flood do pacote.

Após o envio dos pacotes OFPT_PACKET_OUT do controlador para o switch, o controlador envia um pacote OFPT_FLOW_MOD, que serve para alterar a configuração dos flows do switch. Neste caso, o controlador adiciona um novo flow com os endereços MAC e IP de origem e destino, bem como o protocolo IP usado. Uma vez que o protocolo ARP não é o mesmo do que o protocolo ICMP, significa que, para o envio do ping, foi necessário enviar dois pacotes OFPT_FLOW_MOD para cada par origem-destino. Uma vez recebidos e processados pelo switch, já foi possível enviar os restantes pacotes ICMP (request e reply) para os hosts, sem que os mesmos passem pelo controlador.

Exercício 4.2

Configuração da rede

Para a correta configuração da rede, foi necessário criar um ficheiro com uma topologia personalizada em Python, que se encontra em anexo (4-2_Topology.py). O comando usado para iniciar o Mininet foi o seguinte:

```
$ sudo mn --custom 4-2_Topology.py --topo mytopo --mac --controller=none
```

Para facilitar a configuração das rotas nos switches, foi usada a flag `--mac` aquando o arranque do Mininet.

Após o arranque da topologia no Mininet, é necessário adicionar os flows nos switches, de forma a efetuar o correto encaminhamento dos pacotes, bem como evitar loops por parte dos pacotes broadcast. Uma vez que apenas iremos usar o protocolo IPv4, foram bloqueados alguns pacotes ICMPv6 transmitidos na rede.

Para adicionar os flows ao Mininet, é necessário introduzir a lista de comandos do ficheiro "4-2_Flow-commands.sh" no terminal do Mininet. Desta forma, passa a ser possível comunicar entre os hosts.

Testes de funcionamento

Para confirmar que a rede se encontra bem configurada, foram feitos testes de velocidade e pings entre os hosts. Começando pelo ping, o resultado do ping efetuado entre o h3 e h1 é o seguinte:

```
mininet> h3 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=41.1 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=20.2 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=20.2 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=20.2 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=20.1 ms

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
rtt min/avg/max/mdev = 20.112/24.404/41.117/8.358 ms
```

Podemos observar que o RTT (round-trip-time) entre os hosts h3 e h1 tem o valor aproximado de 20 ms (exceto o primeiro ping, que tem um valor superior), uma vez que a rota h3-h1 passa pelo link s3-s1, que tem um delay de 10 ms configurado, pelo que, multiplicando por 2 (pois o RTT é composto pelo envio do pacote para o h1, e pelo envio da resposta, até à receção por parte do h3), temos um delay de 20 ms adicionado propositadamente. Em relação ao primeiro ping, temos de somar ainda o intervalo de tempo do envio do pacote ARP request, e a correspondente resposta do h1, que dá um total de mais 20 ms adicionados "artificialmente" ao RTT, originando assim o valor de 41.1 ms.

Observando o RTT entre o h4 e o h1, percebemos que a rota é diferente (não passa pelo link s3-s1), pois o valor é bastante mais baixo do que o ping entre o h3 e h1, e, apesar da rota ter mais "saltos" (s3-s2-s1), nenhum dos links possui atrasos maiores do que os 10 ms que existem no link s3-s1:

```
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=0.828 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.115 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.124 ms
64 bytes from 10.0.0.1: icmp_seq=4 ttl=64 time=0.142 ms
64 bytes from 10.0.0.1: icmp_seq=5 ttl=64 time=0.120 ms

--- 10.0.0.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4093ms
rtt min/avg/max/mdev = 0.115/0.265/0.828/0.282 ms
```

Olhando agora para a capacidade máxima de cada link, ao analisar a velocidade máxima alcançada na comunicação entre o h3 e h1, conseguimos obter a velocidade máxima correspondente ao link intermédio com menor velocidade. Uma vez que o link s3-s1 possui uma velocidade restrita a 100 Mb/s, enquanto que os restantes links (h3-s3, h1-s1) possuem uma velocidade máxima de 1000 Mb/s, significa que, ao calcular a velocidade máxima da ligação entre os hosts, estamos

realmente a obter a velocidade do link s3-s1. Para tal, é usado o comando "iperf h3 h1", que deu o resultado seguinte:

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

Podemos observar que a velocidade se aproxima dos 100 Mb/s máximos que permite a ligação entre o s3 e o s1, pelo que se confirma a restrição de velocidade imposta. Já ao observar a velocidade máxima da ligação entre o h4 e o h1, o resultado foi o seguinte:

```
mininet> iperf h4 h1
*** Iperf: testing TCP bandwidth between h4 and h1
*** Results: ['943 Mbits/sec', '946 Mbits/sec']
```

A velocidade aproxima-se dos 1000 Mb/s, pelo que podemos confirmar que a velocidade máxima entre o h4 e o h1 é de 1000 Mb/s, no entanto, um dos links que compõem a rota pode possuir uma velocidade máxima superior, pelo que se torna imperativo testar cada um dos links separadamente:

```
mininet> iperf h4 h2
*** Iperf: testing TCP bandwidth between h4 and h2
*** Results: ['937 Mbits/sec', '939 Mbits/sec']
mininet> iperf h2 h1
*** Iperf: testing TCP bandwidth between h2 and h1
*** Results: ['946 Mbits/sec', '950 Mbits/sec']
```

Os resultados comprovam que, em cada link a velocidade encontra-se limitada em ambos os links (s2-s3 e s1-s2).

Para verificar que os pacotes efetuam o percurso pretendido, foram efetuadas capturas no Wireshark. Começando pelo link s3-s1, podemos ver que, ao efetuar o ping entre o h3 e o h1, os pacotes ICMP request e reply passam pelo link, cumprindo assim a rota determinada:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request id=0x651c, seq=1/256, ttl=64 (reply in 2)
2	0.010304891	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply id=0x651c, seq=1/256, ttl=64 (request in 1)
3	1.001091141	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request id=0x651c, seq=2/512, ttl=64 (reply in 4)
4	1.011276019	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply id=0x651c, seq=2/512, ttl=64 (request in 3)
5	2.002617730	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request id=0x651c, seq=3/768, ttl=64 (reply in 6)
6	2.012779647	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply id=0x651c, seq=3/768, ttl=64 (request in 5)
7	3.003419076	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request id=0x651c, seq=4/1024, ttl=64 (reply in 8)
8	3.013548245	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply id=0x651c, seq=4/1024, ttl=64 (request in 7)
9	4.004411920	10.0.0.3	10.0.0.1	ICMP	98	Echo (ping) request id=0x651c, seq=5/1280, ttl=64 (reply in 10)
10	4.014532658	10.0.0.1	10.0.0.3	ICMP	98	Echo (ping) reply id=0x651c, seq=5/1280, ttl=64 (request in 9)

Observando agora a captura de pacotes no link s3-s2, podemos constatar que todos os pacotes com origem no h4 (destino h1 e h2) passam por esse link:

15	96.386860342	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request id=0x65e3, seq=1/256, ttl=64 (reply in 16)
16	96.387160626	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply id=0x65e3, seq=1/256, ttl=64 (request in 15)
17	97.392687057	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request id=0x65e3, seq=2/512, ttl=64 (reply in 18)
18	97.392730769	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply id=0x65e3, seq=2/512, ttl=64 (request in 17)
19	98.767259523	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) request id=0x65e8, seq=1/256, ttl=64 (reply in 20)
20	98.767488943	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) reply id=0x65e8, seq=1/256, ttl=64 (request in 19)
21	99.796663031	10.0.0.4	10.0.0.2	ICMP	98	Echo (ping) request id=0x65e8, seq=2/512, ttl=64 (reply in 22)
22	99.796698805	10.0.0.2	10.0.0.4	ICMP	98	Echo (ping) reply id=0x65e8, seq=2/512, ttl=64 (request in 21)

Por último, as capturas do link s1-s2 confirmam que os pacotes vindos do h4 e do h2, e com destino no h1, passam por este link:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x6693, seq=1/256, ttl=64 (reply in 2)
2	0.000325355	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x6693, seq=1/256, ttl=64 (request in 1)
3	1.025251542	10.0.0.2	10.0.0.1	ICMP	98	Echo (ping) request id=0x6693, seq=2/512, ttl=64 (reply in 4)
4	1.025309742	10.0.0.1	10.0.0.2	ICMP	98	Echo (ping) reply id=0x6693, seq=2/512, ttl=64 (request in 3)
5	3.626191664	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request id=0x6697, seq=1/256, ttl=64 (reply in 6)
6	3.626308492	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply id=0x6697, seq=1/256, ttl=64 (request in 5)
7	4.645287862	10.0.0.4	10.0.0.1	ICMP	98	Echo (ping) request id=0x6697, seq=2/512, ttl=64 (reply in 8)
8	4.645327540	10.0.0.1	10.0.0.4	ICMP	98	Echo (ping) reply id=0x6697, seq=2/512, ttl=64 (request in 7)

Assim, confirma-se o correto funcionamento da topologia solicitada no enunciado.