



Universidade do Minho
Departamento de Informática

Trabalho Prático 2

Redes de Computadores
Grupo 126

Beatriz Rodrigues (*a93230*) Francisco Neves (*a93202*)
Guilherme Fernandes(*a93216*)

Abril 2022

Parte I

1.1. Exercício 1

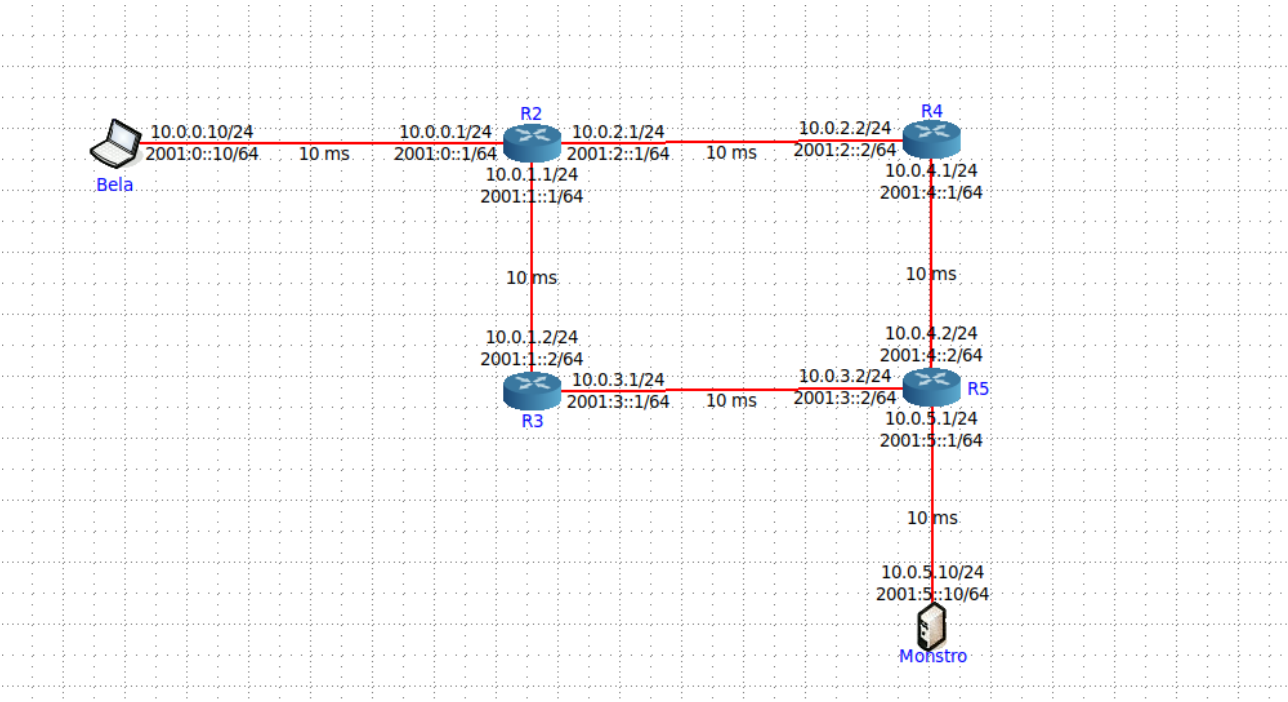


Figura 1.1: Topologia no *CORE*

1.1.a. Active o wireshark ou o tcpdump no host Bela. Numa shell de Bela execute o comando traceroute -I para o endereço IP do Monstro.

No.	Time	Source	Destination	Protocol	Length	Info
3	2.460565378	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=1/256, ttl=1 (no response found!)
4	2.460519802	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=2/512, ttl=1 (no response found!)
5	2.460524261	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=3/768, ttl=1 (no response found!)
6	2.460529554	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=4/1024, ttl=2 (no response found!)
7	2.460533594	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=5/1280, ttl=2 (no response found!)
8	2.460537627	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=6/1536, ttl=2 (no response found!)
9	2.460542786	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=7/1792, ttl=3 (no response found!)
10	2.460546804	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=8/2048, ttl=3 (no response found!)
11	2.460550818	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=9/2304, ttl=3 (no response found!)
12	2.460555923	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=10/2560, ttl=4 (reply in 37)
13	2.460560105	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=11/2816, ttl=4 (reply in 38)
14	2.460564175	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=12/3072, ttl=4 (reply in 39)
15	2.460569268	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=13/3328, ttl=5 (reply in 40)
16	2.460573240	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=14/3584, ttl=5 (reply in 41)
17	2.460577431	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=15/3840, ttl=5 (reply in 42)
18	2.460582845	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=16/4096, ttl=6 (reply in 43)
19	2.483802430	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
20	2.483806071	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
21	2.483808617	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
22	2.484487244	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=17/4352, ttl=6 (reply in 44)
23	2.484495040	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=18/4608, ttl=6 (reply in 45)
24	2.484500290	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=19/4864, ttl=7 (reply in 46)
25	2.511330586	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
26	2.511335868	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
27	2.511336856	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
28	2.511843054	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=20/5120, ttl=7 (reply in 47)
29	2.511852185	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=21/5376, ttl=7 (reply in 48)
30	2.511858244	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=22/5632, ttl=8 (reply in 49)
31	2.533920328	10.0.3.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
32	2.533923819	10.0.3.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
33	2.533924311	10.0.3.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
34	2.534149353	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=23/5888, ttl=8 (reply in 50)
35	2.534154882	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=24/6144, ttl=8 (reply in 51)
36	2.534158511	10.0.0.10	10.0.5.10	ICMP	74	Echo (ping) request id=0x0025, seq=25/6400, ttl=9 (reply in 52)
37	2.557886918	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=10/2560, ttl=61 (request in 12)
38	2.557895753	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=11/2816, ttl=61 (request in 13)
39	2.557897593	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=12/3072, ttl=61 (request in 14)
40	2.557899475	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=13/3328, ttl=61 (request in 15)
41	2.557901179	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=14/3584, ttl=61 (request in 16)
42	2.557902917	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=15/3840, ttl=61 (request in 17)
43	2.557904700	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=16/4096, ttl=61 (request in 18)
44	2.578418749	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=17/4352, ttl=61 (request in 22)
45	2.578428839	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=18/4608, ttl=61 (request in 23)
46	2.578431022	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=19/4864, ttl=61 (request in 24)
47	2.599940982	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=20/5120, ttl=61 (request in 28)
48	2.599950625	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=21/5376, ttl=61 (request in 29)
49	2.599952490	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=22/5632, ttl=61 (request in 30)
50	2.629278802	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=23/5888, ttl=61 (request in 34)
51	2.629288776	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=24/6144, ttl=61 (request in 35)
52	2.629290612	10.0.5.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0025, seq=25/6400, ttl=61 (request in 36)

Figura 1.2: Pacotes capturados pelo *wireshark*

1.1.b. Registe e analise o tráfego ICMP enviado pelo sistema Bela e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

O *traceroute* é um *software* que permite estimar o número de saltos que um pacote dará na rede até encontrar o seu destino. Para isso, ativando a flag **-I**, ele irá enviar pacotes ICMP *echos* e incrementado o *time to live* dos pacotes, ou seja, o número máximo de saltos que estes podem dar. Desta forma, quando um pacote possuir um *time to live* menor do que o necessário, o mesmo não será capaz de chegar ao destino pretendido. Assim, será possível para um emissor descobrir o número mínimo de saltos que o pacote terá de tomar para chegar ao destino pretendido.

Tendo em conta que o *traceroute* é executado no nó Bela para o IP do nó Monstro, através da topologia podemos verificar que o número mínimo de saltos para chegar a esse nó é de 4, logo, todos os pacotes que sejam enviados com um *ttl* menor que 4 não deverão chegar ao destino.

Como podemos ver pela figura representativa dos pacotes capturados pelo *Wireshark*, verifica-se que, conforme esperado, isso é o que de facto acontece. Além disso, é ainda importante referir que são enviados diferentes pacotes com *ttls* diferentes devido ao tempo de propagação e, consequente *delay*, de 10 ms entre os *links* da rede.

Assim sendo, verifica-se que os pacotes que apresentam um valor de *tll* menor que 4 são descartados (*time-to-live exceeded* e que, aos que lhe seguem é recebida como resposta *Echo (ping) reply*.

1.1.c. Qual deve ser o valor inicial mínimo do campo TTL para alcançar o servidor Monstro ? Verifique na prática que a sua resposta está correta.

Tendo em conta o funcionamento do *traceroute*, conforme explicado anteriormente, podemos verificar que o valor inicial mínimo do campo TTL para alcançar o servidor Monstro é de 4. Isto porque, tal como podemos verificar na topologia anteriormente representada, serão necessários, pelo menos, quatro saltos a partir do *host* Bela para alcançarmos o servidor *Monstro*.

De forma a verificar esta premissa, podemos consultar as respostas às alíneas anteriores.

1.1.d. Calcule o valor médio do tempo de ida-e-volta (RTT - Round-Trip Time) obtido no acesso ao servidor. Para melhorar a média, poderá alterar o número pacotes de prova com a opção -q.

Bela \$ `traceroute -I 10.0.5.10 -q 10`

`traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets`

```

1  10.0.0.1 (10.0.0.1)  143.775 ms  143.733 ms  143.730 ms  143.727 ms  143.726 ms
   143.724 ms * * * *
2  10.0.1.2 (10.0.1.2)  165.612 ms  165.610 ms  165.610 ms  165.609 ms  165.607 ms
   165.606 ms * * * *
3  10.0.4.2 (10.0.4.2)  197.135 ms  197.125 ms  177.875 ms  177.837 ms  177.826 ms
   177.816 ms * * * *
4  10.0.5.10 (10.0.5.10)  148.142 ms  148.133 ms  148.122 ms  148.113 ms  99.945 ms
   99.918 ms 99.909 ms 99.901 ms 108.759 ms 108.735 ms
```

$$RTT = \frac{\sum_{i=0}^{10} RTT(pacote_i)}{10} ms \equiv RTT = 120.9077 ms \quad (1.1)$$

1.1.e. O valor médio do atraso num sentido (One-Way Delay) poderia ser calculado com precisão dividindo o RTT por dois? O que torna difícil o cálculo desta métrica?

Não, o cálculo do *One-Way Delay* dessa forma pecaria por se revelar bastante inadequado e, por diversas vezes, bastante impreciso.

Esta imprecisão poderá surgir devido a uma diferença latente entre o tempo de ida e o tempo de volta de um pacote. Tal atraso poderia surgir devido a, por exemplo, algum problema ou congestionamento da rede num dado momento.

Tendo em conta que os caminhos de ida e de volta poderiam apresentar valores muito díspares, o *One-Way Delay* poderia apresentar valores enganosos. No entanto, o cálculo deste valor revela-se bastante difícil pois, não podemos, em tempo real, saber o tempo que o pacote demorou a chegar de um ponto ao outro da rede e, através deste método simplista, poderíamos obter valores incorretos e imprecisos.

1.2. Exercício 2

1.2.a. Qual é o endereço IP da interface ativa do seu computador?

Tendo em conta o pacote capturado após ser efetuado `tracert -I marco.uminho.pt`, podemos compreender que o endereço IP da interface ativa do computador é aquele representado no campo *Source*, visto que será a partir deste que serão enviados os pacotes **ICMP**. Assim, temos que o endereço IP da interface ativa do nosso computador é 172.26.122.70

No.	Time	Source	Destination	Protocol	Length	Info
5	0.003348091	172.26.122.70	193.136.9.240	ICMP	74	Echo (ping) request id=0x0001, seq=1/256, ttl=1 (no response found!)
Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlo1, id 0						
Ethernet II, Src: IntelCor_fa:99:ac (94:b8:6d:fa:99:ac), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)						
Internet Protocol Version 4, Src: 172.26.122.70, Dst: 193.136.9.240						
Internet Control Message Protocol						

Figura 1.3: Primeira mensagem ICMP capturada

1.2.b. Qual é o valor do campo protocolo? O que permite identificar?

O valor do campo protocolo é 1, assim, permite-nos identificar que o tipo de pacote capturado é um pacote pertencente ao protocolo **ICMP**.

▶	Frame 5: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlo1, id 0
▶	Ethernet II, Src: IntelCor_fa:99:ac (94:b8:6d:fa:99:ac), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼	Internet Protocol Version 4, Src: 172.26.122.70, Dst: 193.136.9.240
	0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
▶	Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
	Total Length: 60
	Identification: 0x9472 (38002)
▶	Flags: 0x00
	...0 0000 0000 0000 = Fragment Offset: 0
▶	Time to Live: 1
	Protocol: ICMP (1)
	Header Checksum: 0x3376 [validation disabled]
	[Header checksum status: Unverified]
	Source Address: 172.26.122.70
	Destination Address: 193.136.9.240
▶	Internet Control Message Protocol

Figura 1.4: Campo do protocolo

1.2.c. Quantos bytes tem o cabeçalho IPv4? Quantos bytes tem o campo de dados (payload) do datagrama? Como se calcula o tamanho do payload?

O cabeçalho IPv4 tem tamanho de 20 *bytes*, conforme podemos ver na figura anterior. Além disso, podemos calcular o tamanho do campo de dados através da seguinte fórmula:

$$payload = total\ length - header\ length \quad (1.2)$$

Assim, temos que o valor do *payload* é de 40 *bytes*.

1.2.d. O datagrama IP foi fragmentado? Justifique.

Não, o datagrama IP não foi fragmentado. Para verificarmos isto, basta verificar o campo *more fragments* do datagrama e verificando que este indica que o pacote não possui qualquer fragmento.

Além disso, outra forma de efetuarmos esta verificação seria visualizando o campo *fragment offset* que, no caso de um datagrama possuir fragmentos, existiria, obrigatoriamente, um pacote capturado com o campo diferente de 0.

```
▼ Internet Protocol Version 4, Src: 172.26.122.70, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ► Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0x9472 (38002)
  ▼ Flags: 0x00
    0... .... = Reserved bit: Not set
    .0... .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
  ► Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0x3376 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.122.70
    Destination Address: 193.136.9.240
```

Figura 1.5: *Flags* de fragmentação

1.2.e. Ordene os pacotes capturados de acordo com o endereço IP fonte (e.g., selecionando o cabeçalho da coluna Source), e analise a sequência de tráfego ICMP gerado a partir do endereço IP atribuído à interface da sua máquina. Para a sequência de mensagens ICMP enviadas pelo seu computador, indique que campos do cabeçalho IP variam de pacote para pacote.

Os campos do cabeçalho dos pacotes IP que variam são os seguintes:

- *Identification* - cada pacote IP terá associado a si um valor de identificação;
- *Header Checksum* - como o cabeçalho dos pacotes varia é também de esperar que o valor do seu *checksum* varie;
- *Time to Live* - do funcionamento do **ICMP** será de compreender que este valor irá variar.

5	0.003348091	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=1/256, ttl=1 (no response found!)
6	0.003381265	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=2/512, ttl=1 (no response found!)
7	0.003394712	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=3/768, ttl=1 (no response found!)
8	0.003409265	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=4/1024, ttl=2 (no response found!)
9	0.003420785	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=5/1280, ttl=2 (no response found!)
10	0.003431983	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=6/1536, ttl=2 (no response found!)
11	0.003445773	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=7/1792, ttl=3 (no response found!)
12	0.003456963	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=8/2048, ttl=3 (no response found!)
13	0.003469409	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=9/2304, ttl=3 (no response found!)
14	0.003483443	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=10/2560, ttl=4 (reply in 27)
15	0.003494553	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=11/2816, ttl=4 (reply in 28)
16	0.003505680	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=12/3072, ttl=4 (reply in 29)
17	0.003520041	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=13/3328, ttl=5 (reply in 30)
18	0.003531415	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=14/3584, ttl=5 (reply in 32)
19	0.003542495	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=15/3840, ttl=5 (reply in 34)
20	0.003556853	172.26.122.70	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=16/4096, ttl=6 (reply in 36)

Figura 1.6: Pacotes capturados ordenados pelo endereço IP fonte

1.2.f. Observa algum padrão nos valores do campo de Identificação do datagrama IP e TTL?

Podemos observar que o valor do campo de identificação do datagrama IP é incrementado em 1 unidade por cada novo pacote.

Além disso, podemos ainda verificar que a cada conjunto de 3 pacotes o valor do *ttl* é aumentado em 1 unidade até receber uma resposta positiva, tal como seria de esperar pelo funcionamento do *traceroute*.

1.2.g. Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP TTL exceeded enviadas ao seu computador. Qual é o valor do campo TTL? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL exceeded enviados ao seu host? Porquê?

21	0.008778684	172.26.254.254	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
22	0.008778954	172.26.254.254	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
23	0.008779074	172.26.254.254	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
24	0.008779194	172.16.2.1	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
25	0.008779313	172.16.2.1	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
26	0.008779424	172.16.2.1	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
27	0.008779540	193.136.9.240	172.26.122.70	ICMP	74 Echo (ping) reply	id=0x0001, seq=10/2560, ttl=61 (request in 14)
28	0.008779652	193.136.9.240	172.26.122.70	ICMP	74 Echo (ping) reply	id=0x0001, seq=11/2816, ttl=61 (request in 15)
29	0.008871868	193.136.9.240	172.26.122.70	ICMP	74 Echo (ping) reply	id=0x0001, seq=12/3072, ttl=61 (request in 16)
30	0.008872033	193.136.9.240	172.26.122.70	ICMP	74 Echo (ping) reply	id=0x0001, seq=13/3328, ttl=61 (request in 17)
31	0.008872260	172.16.115.252	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
32	0.009109445	193.136.9.240	172.26.122.70	ICMP	74 Echo (ping) reply	id=0x0001, seq=14/3584, ttl=61 (request in 18)
33	0.009109679	172.16.115.252	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	
34	0.009109898	193.136.9.240	172.26.122.70	ICMP	74 Echo (ping) reply	id=0x0001, seq=15/3840, ttl=61 (request in 19)
35	0.009110126	172.16.115.252	172.26.122.70	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)	

Figura 1.7: Pacotes capturados ordenados pelo endereço de destino

O valor do campo *ttl* dos 3 primeiros pacotes é de 255, dos 3 seguintes é de 254 e dos 3 últimos é de 253.

Assim, podemos verificar que o valor não permanece constante nas mensagens de resposta ICMP TTL *exceeded*, isto deve-se ao facto do *ttl* diminuir em 1 unidade a cada salto.

É ainda interessante referir que cada *router* irá ter um valor de 256 para este campo em cada pacote que envia, no entanto, com os saltos presentes no caminho entre o *router* e a máquina destino, este valor diminui.

1.3. Exercício 3

No.	Time	Source	Destination	Protocol	Length	Info
33	4.980334284	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no response found!)
36	4.980353112	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=2/512, ttl=1 (no response found!)
39	4.980369838	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=3/768, ttl=1 (no response found!)
42	4.980389544	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=4/1024, ttl=2 (no response found!)
45	4.980406783	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=5/1280, ttl=2 (no response found!)
48	4.980423609	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=6/1536, ttl=2 (no response found!)
51	4.980441484	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=7/1792, ttl=3 (no response found!)
54	4.980458131	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=8/2048, ttl=3 (no response found!)
57	4.980475315	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=9/2304, ttl=3 (no response found!)
60	4.980494115	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=10/2560, ttl=4 (reply in 100)
63	4.980516127	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=11/2816, ttl=4 (reply in 103)
66	4.980525913	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=12/3072, ttl=4 (reply in 106)
69	4.980543172	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=13/3328, ttl=5 (reply in 109)
72	4.980560345	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=14/3584, ttl=5 (reply in 112)
75	4.980581434	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=15/3840, ttl=5 (reply in 115)
78	4.980598808	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=16/4096, ttl=6 (reply in 118)
79	4.983432436	172.26.254.254	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
82	4.983782573	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=17/4352, ttl=6 (reply in 121)
83	4.992098751	172.26.254.254	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
86	4.992217152	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=18/4608, ttl=6 (reply in 124)
89	4.993386735	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=19/4864, ttl=7 (reply in 129)
90	4.996746969	172.26.254.254	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
91	5.001539513	172.16.2.1	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
92	5.001539765	172.16.2.1	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
94	5.002125426	172.16.2.1	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
95	5.002125721	172.16.115.252	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
96	5.004232018	172.16.115.252	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
97	5.004423348	172.16.115.252	172.26.122.70	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
100	5.009995684	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=10/2560, ttl=61 (request in 60)
103	5.012346786	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=11/2816, ttl=61 (request in 63)
106	5.012347143	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=12/3072, ttl=61 (request in 66)
109	5.015372718	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=13/3328, ttl=61 (request in 69)
112	5.015373961	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=14/3584, ttl=61 (request in 72)
115	5.016924577	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=15/3840, ttl=61 (request in 75)
118	5.016925007	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=16/4096, ttl=61 (request in 78)
121	5.017015903	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=17/4352, ttl=61 (request in 82)
124	5.017771687	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=18/4608, ttl=61 (request in 86)
129	5.018418227	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=19/4864, ttl=61 (request in 89)
141	5.029378092	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=20/5120, ttl=7 (reply in 148)
144	5.029447633	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=21/5376, ttl=7 (reply in 153)
148	5.033377936	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=20/5120, ttl=61 (request in 141)
153	5.034873971	193.136.9.240	172.26.122.70	ICMP	1180	Echo (ping) reply id=0x0004, seq=21/5376, ttl=61 (request in 144)

Figura 1.8: Mensagens ICMP capturadas ordenada pelo tempo de captura

1.3.a. Localize a primeira mensagem ICMP. Porque é que houve necessidade de fragmentar o pacote inicial?

Tendo em conta o MTU (*Maximum transmission unit*) da *ethernet*, o tamanho máximo de um pacote transferido na rede é de 1500 *bytes*, logo é necessária a fragmentação dos pacotes ICMP (4126 *bytes*)

31	4.980316447	172.26.122.70	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=c705) [Reassembled in #33]
32	4.980330252	172.26.122.70	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=c705) [Reassembled in #33]
33	4.980334284	172.26.122.70	193.136.9.240	ICMP	1180	Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no response found!)

Figura 1.9: Fragmentação do primeiro pacote

1.3.b. Imprima o primeiro fragmento do datagrama IP segmentado. Que informação no cabeçalho indica que o datagrama foi fragmentado? Que informação no cabeçalho IP indica que se trata do primeiro fragmento? Qual é o tamanho deste datagrama IP?

De forma a compreendermos se o datagrama IP foi fragmentado, podemos verificar o valor da flag *more fragments* que está dada como *set*, logo indica que o pacote possui mais fragmentos.

Podemos ainda deduzir que se trata do primeiro fragmento devido ao valor do *fragment offset* que, neste caso, é 0, logo, indica que é o fragmento inicial do pacote.


```

No.      Time      Source      Destination      Protocol Length Info
 31 4.980316447 172.26.122.70 193.136.9.240    IPv4      1514    Fragmented IP protocol (proto=ICMP 1, off=0,
ID=c705) [Reassembled in #33]
Frame 31: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlo1, id 0
Ethernet II, Src: IntelCor_fa:99:ac (94:b8:6d:fa:99:ac), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.122.70, Dst: 193.136.9.240
 0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0xc705 (50949)
Flags: 0x20, More fragments
 0... .... = Reserved bit: Not set
.0... .... = Don't fragment: Not set
..1. .... = More fragments: Set
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 1
Protocol: ICMP (1)
Header Checksum: 0xdb42 [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.26.122.70
Destination Address: 193.136.9.240
[Reassembled IPv4 in frame: 33]

```

Figura 1.10: Informação do primeiro fragmento da datagrama IP

1.3.c. Imprima o segundo fragmento do datagrama IP original. Que informação do cabeçalho IP indica que não se trata do 1º fragmento? Há mais fragmentos? O que nos permite afirmar isso?

Tendo em conta o valor do *fragment offset* ser diferente de 0, podemos concluir, que este não se trata do primeiro fragmento do datagrama.

Podemos ainda compreender que este datagrama possui mais fragmentos, pois, a flag *more fragments* possui o valor *set*, assim, indica-nos a existência de mais fragmentos relativos ao datagrama.

```

No.      Time      Source      Destination      Protocol Length Info
 32 4.980330252 172.26.122.70 193.136.9.240    IPv4      1514    Fragmented IP protocol (proto=ICMP 1,
off=1480, ID=c705) [Reassembled in #33]
Frame 32: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlo1, id 0
Ethernet II, Src: IntelCor_fa:99:ac (94:b8:6d:fa:99:ac), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.122.70, Dst: 193.136.9.240
 0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0xc705 (50949)
Flags: 0x20, More fragments
 0... .... = Reserved bit: Not set
.0... .... = Don't fragment: Not set
..1. .... = More fragments: Set
...0 0101 1100 1000 = Fragment Offset: 1480
Time to Live: 1
Protocol: ICMP (1)
Header Checksum: 0xda89 [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.26.122.70
Destination Address: 193.136.9.240
[Reassembled IPv4 in frame: 33]

```

Figura 1.11: Informação do segundo fragmento da datagrama IP

1.3.d. Quantos fragmentos foram criados a partir do datagrama original?

Conforme podemos ver em *a)*, podemos afirmar que o datagrama originou 3 fragmentos.

É ainda importante referir que o último fragmento possui o valor *not set* na *flag more fragments*, permitindo-nos, dessa forma, compreender que este se trata do último fragmento do datagrama.

Por fim, é ainda relevante reparar que apenas o último fragmento se apresenta como um datagrama ICMP, estando os restantes como datagramas IPv4.

```
No.      Time      Source      Destination      Protocol Length Info
  33 4.980334284 172.26.122.70 193.136.9.240    ICMP      1180    Echo (ping) request id=0x0004, seq=1/256,
ttl=1 (no response found!)
Frame 33: 1180 bytes on wire (9440 bits), 1180 bytes captured (9440 bits) on interface wlo1, id 0
Ethernet II, Src: IntelCor_fa:99:ac (94:b8:6d:fa:99:ac), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.122.70, Dst: 193.136.9.240
 0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1166
Identification: 0xc705 (50949)
Flags: 0x01
 0... .... = Reserved bit: Not set
 .0.. .... = Don't fragment: Not set
 ..0. .... = More fragments: Not set
...0 1011 1001 0000 = Fragment Offset: 2960
Time to Live: 1
Protocol: ICMP (1)
Header Checksum: 0xfble [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.26.122.70
Destination Address: 193.136.9.240
[3 IPv4 Fragments (4106 bytes): #31(1480), #32(1480), #33(1146)]
Internet Control Message Protocol
```

Figura 1.12: Informação do último fragmento da datagrama IP

1.3.e. Indique, resumindo, os campos que mudam no cabeçalho IP entre os diferentes fragmentos, e explique a forma como essa informação permite reconstruir o datagrama original.

Entre os diferentes fragmentos, existem campos que alteram o seu valor e outros que permanecem inalterados para que, no destino dos diversos fragmentos possa ser reconstruída a datagrama original.

Assim, podemos verificar que os seguintes campos apresentam valores diferentes consoante o fragmento em que se encontram:

- *Total Length* - o valor do comprimento do último fragmento poderá ser diferente daqueles que o precedem;
- *Fragment Offset* - o valor do *offset* de cada fragmento será diferente, desta forma, é possível que no destino dos fragmentos estes sejam reordenados de forma a reconstruir a datagrama original;
- *Header Checksum* - o valor do *checksum* do cabeçalho irá variar, visto que, o próprio cabeçalho dos fragmentos será diferente;
- *More Fragments* - o valor da flag *more fragments* irá estar como *not set* no último fragmento e como *set* nos fragmentos que o precedem. Desta forma, o destino poderá compreender qual dos fragmentos será último.

1.3.f. Verifique o processo de fragmentação através de um processo de cálculo.

De forma a verificarmos o processo de fragmentação através de um processo de cálculo, temos de somar os tamanhos dos diferentes fragmentos e subtrair a esse valor o tamanho total dos cabeçalhos dos diferentes fragmentos.

Isto surge, devido à necessidade de criar novos cabeçalhos para cada fragmento o que, em diversas ocasiões, poderá provocar um *overhead*.

Assim, temos que:

$$\sum_{i=1}^{n \text{ fragmentos}} size(fragment_i) - size(ip_header(fragment_i)) \quad (1.3)$$

Ou seja, tendo em conta o comando passado ao *traceroute* em que é indicado que o tamanho do pacote desejado é de 4126 *bytes* e que a esse valor ainda serão retirados 20 *bytes* para o *header*, temos que:

$$1500 - 20 + 1500 - 20 + 1166 - 20 = 4126 - 20 \equiv True \quad (1.4)$$

1.3.g. Escreva uma expressão lógica que permita detetar o último fragmento correspondente ao datagrama original.

$$datagram.more_fragments == 0 \wedge datagram.id == datagram_original.id \wedge datagram.offset \neq 0 \quad (1.5)$$

Parte II

2.1. Exercício 1

- 2.1.a. Indique que endereços IP e máscaras de rede foram atribuídos pelo CORE a cada equipamento. Para simplificar, pode incluir uma imagem que ilustre de forma clara a topologia definida e o endereçamento usado.

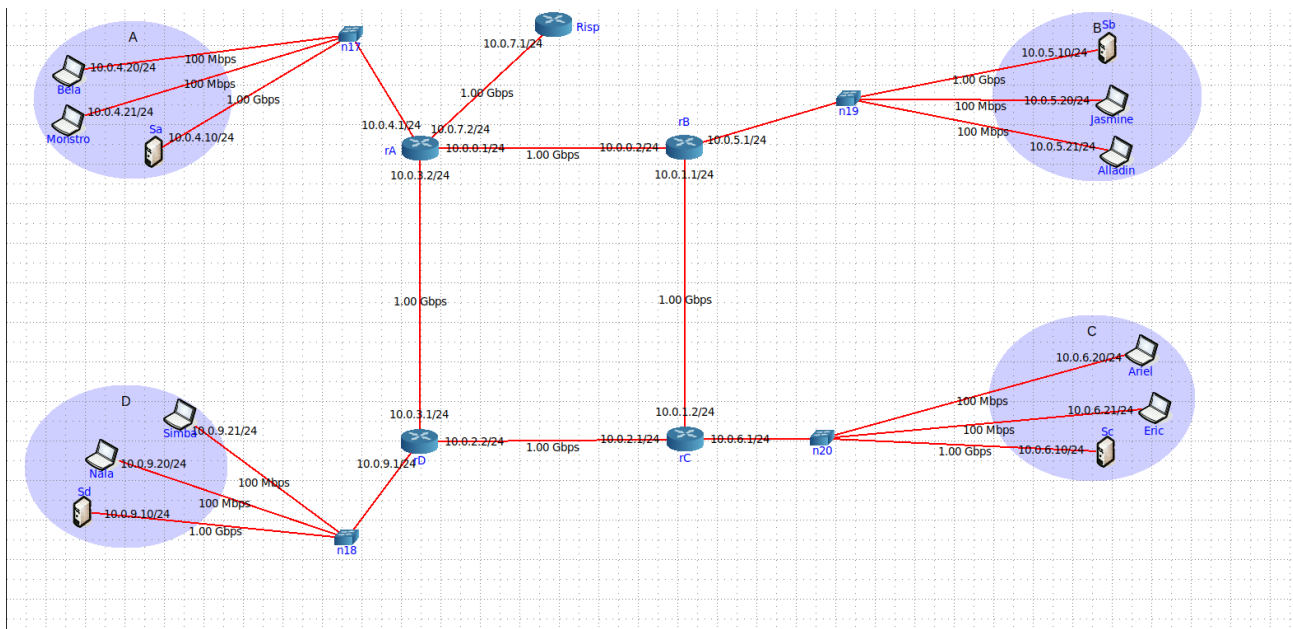


Figura 2.13: Topologia da Rede

Na imagem acima podemos verificar os endereços IP e as máscaras de rede atribuídas pelo CORE a cada equipamento. Podemos ainda verificar que a máscara \24 é correspondente a 255.255.255.0.

- 2.1.b. Tratam-se de endereços públicos ou privados? Porquê?

Tratam-se de endereços privados, uma vez que pertencem à classe B.

- 2.1.c. Porque razão não é atribuído um endereço IP aos switches?

Switches não necessitam de um endereço IP.

Estes são dispositivos da segunda camada, responsabilizam-se por distribuir os pacotes pelos endereços MAC de cada dispositivo, ao contrário de *routers* que o fazem pelos endereços IP.

2.1.d. Usando o comando ping certifique-se que existe conectividade IP interna a cada departamento (e.g. entre um laptop e o servidor respectivo).

```

root@Bela:/tmp/pycore.38127/Bela.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_seq=1 ttl=64 time=0.217 ms
^C
--- 10.0.4.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.217/0.217/0.217/0.000 ms
root@Bela:/tmp/pycore.38127/Bela.conf#

root@Jasmine:/tmp/pycore.38127/Jasmine.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=64 time=0.768 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=0.211 ms
^C
--- 10.0.5.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1032ms
rtt min/avg/max/mdev = 0.211/0.489/0.768/0.278 ms
root@Jasmine:/tmp/pycore.38127/Jasmine.conf#

root@Ariel:/tmp/pycore.38127/Ariel.conf# ping 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=64 time=0.174 ms
^C
--- 10.0.6.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.174/0.174/0.174/0.000 ms
root@Ariel:/tmp/pycore.38127/Ariel.conf#

root@Simba:/tmp/pycore.38127/Simba.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=64 time=3.80 ms
^C
--- 10.0.9.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 3.795/3.795/3.795/0.000 ms
root@Simba:/tmp/pycore.38127/Simba.conf#

```

Figura 2.14: Topologia da Rede

2.1.e. Execute o número mínimo de comandos ping que lhe permite verificar a existência de conetividade IP entre departamentos.

Como sabendo que existindo conetividade *IP* de A para B, então também existe de B para A, o número mínimo de comandos ping necessários para verificar a existencia de conetividade entre todos os departamentos é 6:

```

A → B (B → A)
A → C (C → A)
A → D (D → A)
B → C (B → C)
B → D (D → B)
C → D (D → C)

```

```
Terminal -
File Edit View Terminal Tabs Help
root@Bela:/tmp/pycore.38127/Bela.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=0.293 ms
^C
--- 10.0.5.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.293/0.293/0.293/0.000 ms
root@Bela:/tmp/pycore.38127/Bela.conf# ping 10.0.6.20
PING 10.0.6.20 (10.0.6.20) 56(84) bytes of data.
64 bytes from 10.0.6.20: icmp_seq=1 ttl=61 time=6.83 ms
^C
--- 10.0.6.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 6.829/6.829/6.829/0.000 ms
root@Bela:/tmp/pycore.38127/Bela.conf# ping 10.0.9.21
PING 10.0.9.21 (10.0.9.21) 56(84) bytes of data.
64 bytes from 10.0.9.21: icmp_seq=1 ttl=62 time=1.95 ms
^C
--- 10.0.9.21 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.953/1.953/1.953/0.000 ms
root@Bela:/tmp/pycore.38127/Bela.conf#
```

Figura 2.15: ping do departamento A para os departamentos B, C e D

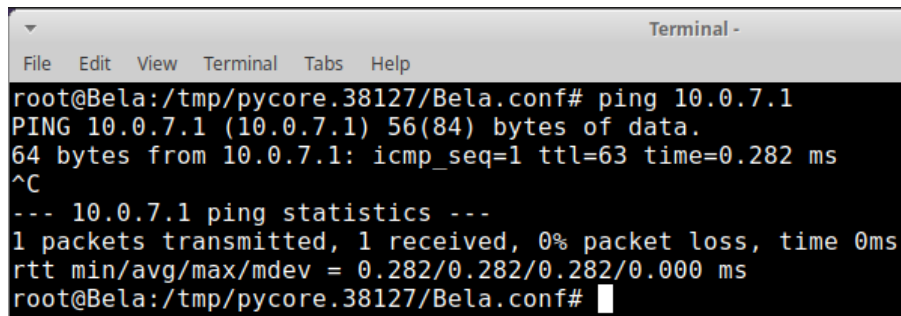
```
Terminal -
File Edit View Terminal Tabs Help
root@Jasmine:/tmp/pycore.38127/Jasmine.conf# ping 10.0.6.20
PING 10.0.6.20 (10.0.6.20) 56(84) bytes of data.
64 bytes from 10.0.6.20: icmp_seq=1 ttl=62 time=2.57 ms
^C
--- 10.0.6.20 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.573/2.573/2.573/0.000 ms
root@Jasmine:/tmp/pycore.38127/Jasmine.conf# ping 10.0.9.21
PING 10.0.9.21 (10.0.9.21) 56(84) bytes of data.
64 bytes from 10.0.9.21: icmp_seq=1 ttl=61 time=0.481 ms
^C
--- 10.0.9.21 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.481/0.481/0.481/0.000 ms
root@Jasmine:/tmp/pycore.38127/Jasmine.conf#
```

Figura 2.16: ping do departamento B para os departamentos C e D

```
Terminal -
File Edit View Terminal Tabs Help
root@Ariel:/tmp/pycore.38127/Ariel.conf# ping 10.0.9.21
PING 10.0.9.21 (10.0.9.21) 56(84) bytes of data.
64 bytes from 10.0.9.21: icmp_seq=1 ttl=62 time=8.73 ms
^C
--- 10.0.9.21 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 8.732/8.732/8.732/0.000 ms
root@Ariel:/tmp/pycore.38127/Ariel.conf#
```

Figura 2.17: ping do departamento C para o departamento D

- 2.1.f. Verifique se existe conectividade IP do portátil *Bela* para o router de acesso R_{ISP} .



```
root@Bela:/tmp/pycore.38127/Bela.conf# ping 10.0.7.1
PING 10.0.7.1 (10.0.7.1) 56(84) bytes of data:
64 bytes from 10.0.7.1: icmp_seq=1 ttl=63 time=0.282 ms
^C
--- 10.0.7.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.282/0.282/0.282/0.000 ms
root@Bela:/tmp/pycore.38127/Bela.conf#
```

Figura 2.18: ping do portátil *Bela* para o router R_{ISP}

2.2. Exercício 2

- 2.2.a. Execute o comando `netstat -rn` por forma a poder consultar a tabela de encaminhamento unicast (IPv4). Inclua no seu relatório as tabelas de encaminhamento obtidas; interprete as várias entradas de cada tabela. Se necessário, consulte o manual respetivo (`man netstat`).

Relativamente ao encaminhamento Unicast do Portátil *Bela*, temos a seguinte tabela de encaminhamento.

```
root@Bela:/tmp/pycore.38127/Bela.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask          Flags      MSS Window  irtt Iface
0.0.0.0          10.0.4.1         0.0.0.0          UG         0 0        0 eth0
10.0.4.0         0.0.0.0         255.255.255.0    U          0 0        0 eth0
```

Por outro lado, quanto à tabela de encaminhamento para o router *A* temos:

```
root@rA:/tmp/pycore.38127/rA.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask          Flags      MSS Window  irtt Iface
10.0.0.0         0.0.0.0         255.255.255.0    U          0 0        0 eth0
10.0.1.0         10.0.0.2        255.255.255.0    UG         0 0        0 eth0
10.0.2.0         10.0.3.1        255.255.255.0    UG         0 0        0 eth1
10.0.3.0         0.0.0.0         255.255.255.0    U          0 0        0 eth1
10.0.4.0         0.0.0.0         255.255.255.0    U          0 0        0 eth2
10.0.5.0         10.0.0.2        255.255.255.0    UG         0 0        0 eth0
10.0.6.0         10.0.0.2        255.255.255.0    UG         0 0        0 eth0
10.0.7.0         0.0.0.0         255.255.255.0    U          0 0        0 eth3
10.0.9.0         10.0.3.1        255.255.255.0    UG         0 0        0 eth1
```

Tendo em conta as tabelas obtidas, podemos verificar que a primeira coluna correspondente a *Destination* é relativa ao destino que um pacote deverá ter, assim, um pacote com um determinado destino, poderá ser reencaminhado por parte do nó em questão, para o próximo nó de forma correta. O próximo nó é representado pelo campo *Gateway* da tabela de encaminhamento

e será para ele que um pacote deverá seguir caminho. O campo *Genmask* é responsável por representar a máscara de rede, sendo, neste caso, dada por 255.255.255.0 que é o equivalente a $\backslash 24$.

Além disso, as *flags* U indicam que a rota encontra-se disponível para uso e as *flags* G indicam que a rota é para um *gateway*.

Por fim, o campo *Iface* permite sabermos as interfaces correspondentes a cada entrada da tabela de encaminhamento.

2.2.b. Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema, por exemplo, `ps -ax` ou equivalente).

Efetuada o comando `ps -ax` no *router* R_A , podemos verificar que um dos processos que se encontra a correr naquele dispositivo é correspondente a um `ospfd`. Sendo que este processo retrata um protocolo de encaminhamento dinâmico, *Open Shortest Path First*, podemos concluir que está a ser utilizado um protocolo de encaminhamento dinâmico.

```

root@rA:/tmp/pycore.38127/rA.conf# ps -ax
  PID TTY          STAT TIME  COMMAND
    1 ?            S     0:00 vnodes -v -c /tmp/pycore.38127/rA -l /tmp/pycore.38
   76 ?            Ss    0:00 /usr/local/sbin/zebra -d
   82 ?            Ss    0:00 /usr/local/sbin/ospf6d -d
   93 ?            Ss    0:01 /usr/local/sbin/ospfd -d
  108 pts/2        Ss    0:00 /bin/bash
  115 pts/2        R+    0:00 ps -ax

```

Figura 2.19: Processos em execução no *Router A*

2.2.c. Admita que, por questões administrativas, a rota por defeito (0.0.0.0 ou default) deve ser retirada definitivamente da tabela de encaminhamento do servidor S_A . Use o comando `route delete` para o efeito. Que implicações tem esta medida para os utilizadores da LEI-RC que acedem ao servidor. Justifique.

```

root@Sa:/tmp/pycore.38127/Sa.conf# route del default
root@Sa:/tmp/pycore.38127/Sa.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@Sa:/tmp/pycore.38127/Sa.conf# ping 10.0.5.10
ping: connect: Network is unreachable
root@Sa:/tmp/pycore.38127/Sa.conf#

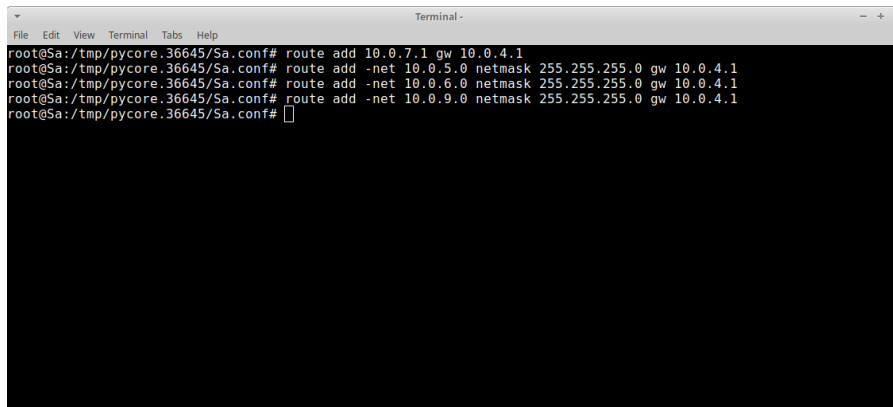
```

Figura 2.20: Remoção da rota *default*

Os restantes utilizadores da rede possuem rotas para o servidor S_A , no entanto, este não consegue responder ao pedido de nodos externos à sua sub-rede pois não possui uma rota para eles.

Para nodos dentro da sua sub rede, o servidor possui o *gateway* 0.0.0.0, ou seja o servidor é o seu próprio *gateway* e utiliza o ARP (*Address Resolution Protocol*) para descobrir os endereços da sua sub-rede.

- 2.2.d. Não volte a repor a rota por defeito. Adicione todas as rotas estáticas necessárias para restaurar a conectividade para o servidor S_A , por forma a contornar a restrição imposta na alínea c). Utilize para o efeito o comando `route add` e registe os comandos que usou.

A terminal window titled "Terminal -" with a menu bar (File, Edit, View, Terminal, Tabs, Help). The prompt is "root@Sa:/tmp/pycore.36645/Sa.conf#". Four commands are entered and executed: "route add 10.0.7.1 gw 10.0.4.1", "route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.4.1", "route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.4.1", and "route add -net 10.0.9.0 netmask 255.255.255.0 gw 10.0.4.1". The prompt returns after each command.

```
root@Sa:/tmp/pycore.36645/Sa.conf# route add 10.0.7.1 gw 10.0.4.1
root@Sa:/tmp/pycore.36645/Sa.conf# route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.4.1
root@Sa:/tmp/pycore.36645/Sa.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.4.1
root@Sa:/tmp/pycore.36645/Sa.conf# route add -net 10.0.9.0 netmask 255.255.255.0 gw 10.0.4.1
root@Sa:/tmp/pycore.36645/Sa.conf#
```

Figura 2.21: Adição das rotas estáticas

Tendo em conta que é pretendido que o servidor S_A seja capaz de comunicar com o exterior, torna-se necessária a adição de entradas na tabela de *routing* que permitam essa comunicação. Assim, podemos prever que será necessária uma entrada na tabela para as diferentes sub-redes e, até mesmo, para o *router* R_{ISP} .

Assim, foram adicionadas as rotas pretendidas. De notar que para o *router* R_{ISP} não é especificada a flag `-net`, visto este não se tratar de uma rede. Por fim, os campos `netmask` e `gw` correspondem à máscara de rede utilizada, que, tal como podemos ver nos endereços IP é dada por `\24`, logo, representada por `255.255.255.0` e ao *gateway* que será sempre `10.0.4.1`, visto ser a interface do *router* a que o servidor se encontra ligado, logo, terá de ser, obrigatoriamente, o próximo salto, respetivamente.

- 2.2.e. Teste a nova política de encaminhamento garantindo que o servidor está novamente acessível, utilizando para o efeito o comando `ping`. Registe a nova tabela de encaminhamento do servidor.

```

root@Sa:/tmp/pycore.36645/Sa.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=0.455 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=9.81 ms
^C
--- 10.0.5.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1029ms
rtt min/avg/max/mdev = 0.455/5.131/9.807/4.676 ms
root@Sa:/tmp/pycore.36645/Sa.conf# ping 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=61 time=9.73 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=61 time=24.8 ms
^C
--- 10.0.6.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1008ms
rtt min/avg/max/mdev = 9.727/17.277/24.828/7.550 ms
root@Sa:/tmp/pycore.36645/Sa.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=62 time=4.12 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=62 time=11.7 ms
^C
--- 10.0.9.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 4.121/7.913/11.705/3.792 ms
root@Sa:/tmp/pycore.36645/Sa.conf# ping 10.0.7.1
PING 10.0.7.1 (10.0.7.1) 56(84) bytes of data.
64 bytes from 10.0.7.1: icmp_seq=1 ttl=63 time=0.208 ms
64 bytes from 10.0.7.1: icmp_seq=2 ttl=63 time=2.91 ms
^C
--- 10.0.7.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1043ms
rtt min/avg/max/mdev = 0.208/1.556/2.905/1.348 ms
root@Sa:/tmp/pycore.36645/Sa.conf# ping 10.0.4.20
PING 10.0.4.20 (10.0.4.20) 56(84) bytes of data.
64 bytes from 10.0.4.20: icmp_seq=1 ttl=64 time=0.301 ms
64 bytes from 10.0.4.20: icmp_seq=2 ttl=64 time=4.87 ms
^C
--- 10.0.4.20 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1030ms
rtt min/avg/max/mdev = 0.301/2.583/4.866/2.282 ms
root@Sa:/tmp/pycore.36645/Sa.conf# █

```

Figura 2.22: Pings entre S1 e nodos da topologia

Segue-se a tabela de encaminhamento.

```

root@Sa:/tmp/pycore.36645/Sa.conf# netstat -rn
Kernel IP routing table

```

Destination	Gateway	Genmask	Flags	MSS Window	irrt	Iface
10.0.4.0	0.0.0.0	255.255.255.0	U	0 0	0	eth0
10.0.5.0	10.0.4.1	255.255.255.0	UG	0 0	0	eth0
10.0.6.0	10.0.4.1	255.255.255.0	UG	0 0	0	eth0
10.0.7.1	10.0.4.1	255.255.255.255	UGH	0 0	0	eth0
10.0.9.0	10.0.4.1	255.255.255.0	UG	0 0	0	eth0

2.3. Exercício 3

- 2.3.a. Considere que dispõe apenas do endereço de rede IP 192.168.XXX.128/25, em que XXX é o decimal correspondendo ao seu número de grupo (PLXX). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo as redes de acesso externo e *backbone* inalteradas), sabendo que o número de departamentos pode vir a aumentar no curto prazo. Atribua endereços às interfaces dos vários sistemas envolvidos. Assuma que todos os endereços de sub-redes são usáveis. Justifique as opções tomadas no planeamento.

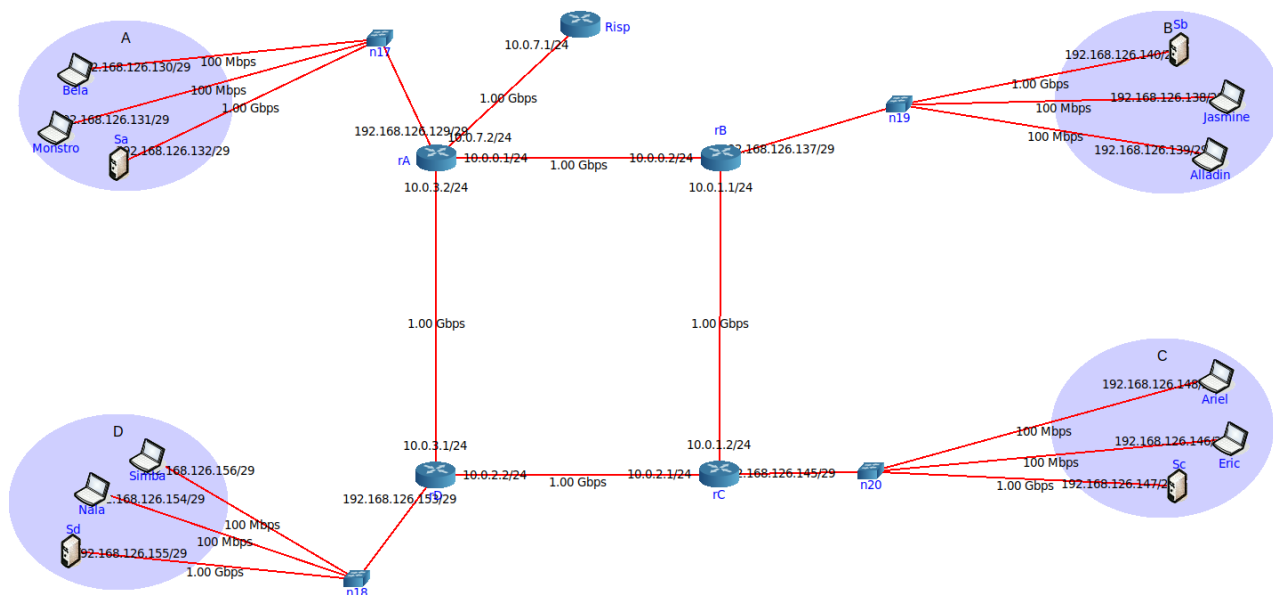


Figura 2.23: Topologia com os novos endereços

Dado o número do grupo (PL126), o endereço da rede IP será 192.168.126.128/25. Em binário teremos então o seguinte endereço 11000000.10101000.01111110.10000000, em que os primeiros 25 *bits* se encontram reservados, logo, podemos utilizar 7 *bits* para atribuição de endereços e definição de sub-redes.

Como não é especificado o aumento esperado do número de departamentos, decidimos utilizar 4 *bits* para *subnetting*, apesar que, uma utilização de 3 *bits* permitiria a definição de *subnetting* para 8 departamentos (2^3). No entanto, visto não sabermos se a expansão do número de departamentos seria de maior dimensão que esta e que o número de *hosts* não é expectável que aumente, optamos pela decisão de aumentar o número de departamentos possíveis, permitindo assim a existência de 16 departamentos.

2.3.b. Qual a máscara de rede que usou (em formato decimal)? Quantos hosts IP pode interligar em cada departamento? Quantos prefixos de sub-rede ficam disponíveis para uso futuro? Justifique.

A máscara de rede utilizada foi \29, que podemos representar por 255.255.255.248, visto que decidimos utilizar 4 dos 7 *bits* disponíveis para *subnetting*. Podemos ligar 8 *hosts* a cada departamento, visto que ficaram 3 *bits* disponíveis para *host* (2^3 , considerando todos os endereços usáveis). Por fim, ficaram disponíveis 12 prefixos de sub-rede para uso futuro, permitindo a inserção de 12 novos departamentos na topologia ($2^4 - 4$, visto já termos 4 prefixos em uso).

2.3.c. Verifique e garanta que a conectividade IP interna na rede local LEI-RC é mantida. No caso de não existência de conectividade, reveja a atribuição de endereços efetuada e eventuais erros de encaminhamento por forma a realizar as correções necessárias. Explique como procedeu.

Não foi necessário efetuar qualquer correção, visto que o *CORE* atualizou as tabelas de *routing* com os novos endereços da rede.

```

root@Bela:/tmp/pycore.36645/Bela.conf# ping 192.168.126.140
PING 192.168.126.140 (192.168.126.140) 56(84) bytes of data.
64 bytes from 192.168.126.140: icmp_seq=1 ttl=62 time=0.141 ms
^C
--- 192.168.126.140 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.141/0.141/0.141/0.000 ms
root@Bela:/tmp/pycore.36645/Bela.conf# ping 192.168.126.148
PING 192.168.126.148 (192.168.126.148) 56(84) bytes of data.
64 bytes from 192.168.126.148: icmp_seq=1 ttl=61 time=0.567 ms
^C
--- 192.168.126.148 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.567/0.567/0.567/0.000 ms
root@Bela:/tmp/pycore.36645/Bela.conf# ping 192.168.126.156
PING 192.168.126.156 (192.168.126.156) 56(84) bytes of data.
64 bytes from 192.168.126.156: icmp_seq=1 ttl=62 time=0.702 ms
^C
--- 192.168.126.156 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.702/0.702/0.702/0.000 ms
root@Bela:/tmp/pycore.36645/Bela.conf# █

```

Figura 2.24: ping de A para os restantes departamentos

```

root@Sb:/tmp/pycore.36645/Sb.conf# ping 192.168.126.148
PING 192.168.126.148 (192.168.126.148) 56(84) bytes of data.
64 bytes from 192.168.126.148: icmp_seq=1 ttl=62 time=0.341 ms
^C
--- 192.168.126.148 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.341/0.341/0.341/0.000 ms
root@Sb:/tmp/pycore.36645/Sb.conf# ping 192.168.126.156
PING 192.168.126.156 (192.168.126.156) 56(84) bytes of data.
64 bytes from 192.168.126.156: icmp_seq=1 ttl=61 time=0.385 ms
^C
--- 192.168.126.156 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.385/0.385/0.385/0.000 ms
root@Sb:/tmp/pycore.36645/Sb.conf# █

```

Figura 2.25: ping de B para os departamentos C e D

```

root@Ariel:/tmp/pycore.36645/Ariel.conf# ping 192.168.126.156
PING 192.168.126.156 (192.168.126.156) 56(84) bytes of data.
64 bytes from 192.168.126.156: icmp_seq=1 ttl=62 time=0.201 ms
^C
--- 192.168.126.156 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.201/0.201/0.201/0.000 ms
root@Ariel:/tmp/pycore.36645/Ariel.conf# █

```

Figura 2.26: ping de C para D

```

root@Simba:/tmp/pycore.36645/Simba.conf# ping 10.0.7.1
PING 10.0.7.1 (10.0.7.1) 56(84) bytes of data.
64 bytes from 10.0.7.1: icmp_seq=1 ttl=62 time=0.490 ms
^C
--- 10.0.7.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.490/0.490/0.490/0.000 ms
root@Simba:/tmp/pycore.36645/Simba.conf# █

```

Figura 2.27: ping de D para R_{ISP}

Conclusões

Com este trabalho, foi possível a consolidação de alguns temas lecionados na unidade curricular de Redes de Computadores.

Relativamente à primeira parte, foi possível compreender como é estruturado um datagrama IP, ao capturarmos os pacotes através do *Wireshark*. Para além disso, foi estudado o comando `traceroute` e as suas potencialidades. Por fim, conseguiu-se aprofundar o conceito de fragmentação e aprender a detetar se um pacote foi, de facto, fragmentado e porquê.

Por outro lado, relativamente à segunda parte, percebeu-se como funciona o endereçamento e encaminhamento IP numa topologia com múltiplos *routers* e *switches*. Devido à existência de múltiplos departamentos, foi ainda possível consolidar o conceito de criação de *subnetting*, ao modificarmos a topologia com uma nova máscara que permitisse a existência dos diversos departamentos e outros novos que pudessem surgir no futuro..