

UNIVERSIDADE DO MINHO

LICENCIATURA/MESTRADO INTEGRADO EM ENGENHARIA
INFORMÁTICA

Redes de Computadores - TP2 (Parte 1)
Grupo 94

Rui Guilherme Monteiro (A93179) Rui Moreira (A93232)
José Pereira (A89596)

Ano Lectivo 2021/2022



Conteúdo

1 Exercícios - Parte I	4
1.1 Exercício 1	4
1.1.1 Alínea a)	4
1.1.2 Alínea b)	5
1.1.3 Alínea c)	6
1.1.4 Alínea d)	6
1.1.5 Alínea e)	6
1.2 Exercício 2	7
1.2.1 Alínea a)	7
1.2.2 Alínea b)	7
1.2.3 Alínea c)	7
1.2.4 Alínea d)	8
1.2.5 Alínea e)	8
1.2.6 Alínea f)	9
1.2.7 Alínea g)	9
1.3 Exercício 3	10
1.3.1 Alínea a)	11
1.3.2 Alínea b)	11
1.3.3 Alínea c)	12
1.3.4 Alínea d)	13
1.3.5 Alínea e)	13
1.3.6 Alínea f)	13
1.3.7 Alínea g)	15
2 Exercícios - Parte II	16
2.1 Exercício 1	16
2.1.1 Alínea a)	16
2.1.2 Alínea b)	17
2.1.3 Alínea c)	17
2.1.4 Alínea d)	18
2.1.5 Alínea e)	19
2.1.6 Alínea f)	20
2.2 Exercício 2	21
2.2.1 Alínea a)	21
2.2.2 Alínea b)	22
2.2.3 Alínea c)	23

2.2.4	Alínea d)	23
2.2.5	Alínea e)	23
2.3	Exercício 3	25
2.3.1	Alínea 1)	25
2.3.2	Alínea 2)	26
2.3.3	Alínea 3)	26
3	Conclusão	29

Capítulo 1

Exercícios - Parte I

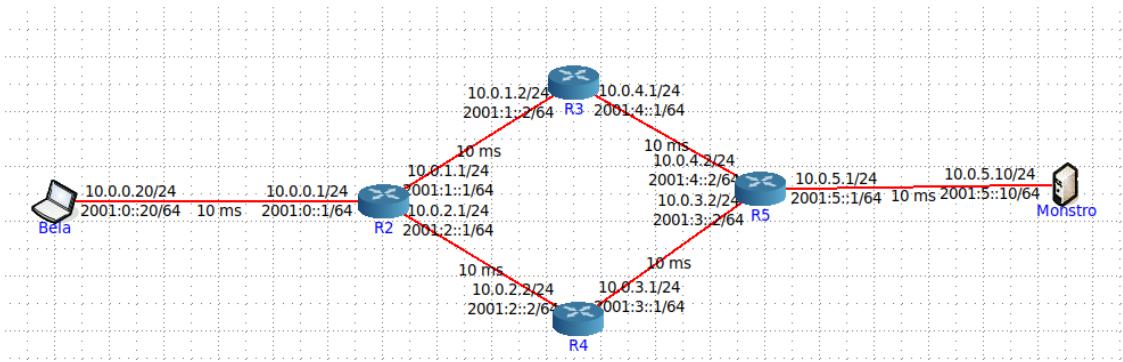


Figura 1.1: Topologia

1.1 Exercício 1

1.1.1 Alínea 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.

```
root@Bela:/tmp/pycore.46699/Bela.conf# traceroute -I 10.0.5.10
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)  50.485 ms  50.444 ms  50.443 ms
 2  10.0.1.2 (10.0.1.2)  71.778 ms  71.778 ms  71.777 ms
 3  10.0.3.2 (10.0.3.2)  132.869 ms  132.869 ms  132.868 ms
 4  10.0.5.10 (10.0.5.10)  217.090 ms  217.091 ms  217.092 ms
root@Bela:/tmp/pycore.46699/Bela.conf#
```

Figura 1.2: Comando traceroute -I

1.1.2 Alínea 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 comportamento foi o esperado. Foram enviados inicialmente com o campo TTL igual 1 e foi sendo incrementado até ser possível estabelecer conexão com sucesso. Os pacotes com TTL 6, 7 e 8 excederam o seu TTL ainda em trânsito, no *router* R2, R3 e R4, respectivamente tendo sido recebido um pacote *Time-to-live exceeded* vindos desses routers, por cada pacote em que foi excedido o TTL.

19	15.054684250	00:00:00_aa:00:01	00:00:00_aa:00:00	ARP	42	10.0.0.1 is at 00:00:00_aa:00:01
20	15.054706671	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response found!)
21	15.054707131	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=2/512, ttl=1 (no response found!)
22	15.054707482	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response found!)
23	15.054707819	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no response found!)
24	15.054707819	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no response found!)
25	15.054708558	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no response found!)
26	15.054709106	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no response found!)
27	15.054709720	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no response found!)
28	15.054710200	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no response found!)
29	15.054710678	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in 54)
30	15.054711162	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in 55)
31	15.054711615	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (reply in 56)
32	15.054711987	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (reply in 57)
33	15.054712322	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (reply in 58)
34	15.054712644	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (reply in 59)
35	15.054712999	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (reply in 60)
36	15.084164952	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
37	15.084172209	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
38	15.084172884	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
39	15.084841193	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=17/4352, ttl=6 (reply in 61)
40	15.084849058	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=18/4608, ttl=6 (reply in 62)
41	15.084852745	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=19/4864, ttl=7 (reply in 63)
42	15.105504246	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
43	15.105511534	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
44	15.105512155	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
45	15.105748339	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=20/5120, ttl=7 (reply in 64)
46	15.105755767	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=21/5376, ttl=7 (reply in 65)
47	15.105759250	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=22/5632, ttl=8 (reply in 66)
48	15.166600102	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
49	15.166607389	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
50	15.166608341	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
51	15.166901995	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=23/5888, ttl=8 (reply in 67)
52	15.166909154	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=24/6144, ttl=8 (reply in 68)
53	15.166912763	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=25/6400, ttl=9 (reply in 69)
54	15.250822925	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=10/2560, ttl=61 (request in 29)

Figura 1.3: Enviado

53	15.166912763	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=25/6400, ttl=9 (reply in 69)
54	15.250822925	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=10/2560, ttl=61 (request in 29)
55	15.250835111	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=11/2816, ttl=61 (request in 30)
56	15.250836664	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=12/3072, ttl=61 (request in 31)
57	15.250838851	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=13/3328, ttl=61 (request in 32)
58	15.250840208	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=14/3584, ttl=61 (request in 33)
59	15.250841612	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=15/3840, ttl=61 (request in 34)
60	15.250843286	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=16/4096, ttl=61 (request in 35)
61	15.250845001	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=17/4352, ttl=61 (request in 39)
62	15.250846469	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=18/4608, ttl=61 (request in 40)
63	15.250848023	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=19/4864, ttl=61 (request in 41)
64	15.250849534	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=20/5120, ttl=61 (request in 45)
65	15.250851019	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=21/5376, ttl=61 (request in 46)
66	15.250852361	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=22/5632, ttl=61 (request in 47)
67	15.295387687	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=23/5888, ttl=61 (request in 51)
68	15.295395003	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=24/6144, ttl=61 (request in 52)
69	15.295395958	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=25/6400, ttl=61 (request in 53)
70	16.100726087	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet

Figura 1.4: Respondido

1.1.3 Alínea 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.

20 15.054706671 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response found!)
21 15.054707131 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=2/512, ttl=1 (no response found!)
22 15.054707482 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response found!)
23 15.054707819 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no response found!)
24 15.054708225 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no response found!)
25 15.054708558 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no response found!)
26 15.054709188 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no response found!)
27 15.054709729 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no response found!)
28 15.054710289 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no response found!)
29 15.054710678 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in 54)
30 15.054711162 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in 55)
31 15.054711615 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (reply in 56)
32 15.054711987 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (reply in 57)
33 15.054712322 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (reply in 58)
34 15.054712644 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (reply in 59)
35 15.054712999 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (reply in 60)
36 15.084164952 10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (time to live exceeded in transit)
37 15.084172209 10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (time to live exceeded in transit)
38 15.084172884 10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (time to live exceeded in transit)
39 15.084841193 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=17/4352, ttl=6 (reply in 61)

Figura 1.5: Tráfego ICMP

Observando a topologia da rede, na Figura 2.21, o valor mínimo esperado do TTL é 4 para um pacote atravessar com sucesso. Como podemos ver na Figura 1.5, até ao TTL=3, os pacotes são descartados ("No response found"), só a partir do TTL=4 é que há a primeira resposta.

1.1.4 Alínea 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.

```
root@Bela:/tmp/pycore.40703/Bela.conf# 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)  21.374 ms  21.363 ms  21.361 ms  21.359 ms  21.358 ms
21.356 ms * * *
 2  10.0.1.2 (10.0.1.2)  42.511 ms  42.510 ms  42.509 ms  42.507 ms  42.507 ms
42.505 ms * * *
 3  10.0.3.2 (10.0.3.2)  62.310 ms  62.308 ms  61.605 ms  61.596 ms  61.594 ms
61.592 ms * * *
 4  10.0.5.10 (10.0.5.10)  84.684 ms  84.682 ms  84.526 ms  84.515 ms  86.264 ms
86.251 ms  86.249 ms  86.247 ms  82.792 ms  82.783 ms
```

Figura 1.6: Comando traceroute com a opção -q

O valor médio do tempo de ida-e-volta obtido no acesso ao servidor obtido foi 84,8993 ms.

1.1.5 Alínea 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?

O cálculo do atraso num sentido (*One-Way Delay*) não pode ser calculado com precisão através

da divisão do RTT por dois, visto que o caminho de ida pode não ser o mesmo que o caminho de volta. Para calculá-lo com precisão, seria necessário informação de filas, tempos de processamento em cada nodo, tempo de propagação entre links, entre outros fatores que o afetariam.

1.2 Exercício 2

1.2.1 Alínea a)

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

Analizando a Figura 1.7, o endereço IP do computador utilizado é 172.26.26.189.

```
v Internet Protocol Version 4, Src: 172.26.26.189, 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: 72
        Identification: 0x84dc (34012)
    > Flags: 0x00
        ...0 0000 0000 0000 = Fragment Offset: 0
    > Time to Live: 1
        Protocol: ICMP (1)
        Header Checksum: 0xa289 [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 172.26.26.189
        Destination Address: 193.136.9.240
```

Figura 1.7: Endereço IP da interface activa do computador

1.2.2 Alínea b)

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

Como podemos ver na Figura 1.7, na parte destacada a verde, o valor do campo protocolo é 1, equivale ao ICMP.

1.2.3 Alínea c)

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

Analizando o pacote apresentado na Figura 1.7, nas partes destacadas a azul, é possível verificar que o campo correspondente ao *Header Length* tem o valor de 20 bytes. De modo a obter o valor do campo de dados (*payload*), subtrai-se o *Header Length* ao *Total Length*, o que equivale a 52 bytes.

1.2.4 Alínea d)

O datagrama IP foi fragmentado? Justifique.

```

> Frame 1: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface en0, id 0
> Ethernet II, Src: Apple_4f:4b:78 (f0:18:98:4f:4b:78), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.26.189, 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: 72
  Identification: 0x84dc (34012)
  < 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: 0xa289 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.26.189
  Destination Address: 193.136.9.240
  > Internet Control Message Protocol

```

Figura 1.8: Flags do Protocolo IP

Na Figura 1.8 verifica-se que, tanto o campo *More Fragments* como o campo *Fragment offset* têm o valor 0, o que confirma que **o datagrama IP não foi fragmentado**.

1.2.5 Alínea 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.

Ordenando os pacotes capturados de acordo com o endereço IP fonte, obtém-se a seguinte figura:

1 0.000000	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=1/256, ttl=1 (no response found!)
3 0.009469	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=2/512, ttl=1 (no response found!)
5 0.011697	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=3/768, ttl=1 (no response found!)
7 0.015392	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=4/1024, ttl=2 (no response found!)
9 0.018244	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=5/1280, ttl=2 (no response found!)
11 0.020241	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=6/1536, ttl=2 (no response found!)
13 0.022262	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=7/1792, ttl=3 (no response found!)
15 0.025727	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=8/2048, ttl=3 (no response found!)
17 0.028526	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=9/2304, ttl=3 (no response found!)
19 0.031353	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=10/2560, ttl=4 (reply in 20)
21 0.034082	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=11/2816, ttl=4 (reply in 22)
23 0.036196	172.26.26.189	193.136.9.240	ICMP	86 Echo (ping) request id=0x84db, seq=12/3072, ttl=4 (reply in 24)

Figura 1.9: Pacotes ordenados de acordo com o endereço IP fonte

Depois de analisar as mensagens ICMP enviadas pelo nosso computador, verificamos que os campos do cabeçalho IP que variam de pacote para pacote são:

- *Identification*
- *Time to live*

- *Header Checksum*

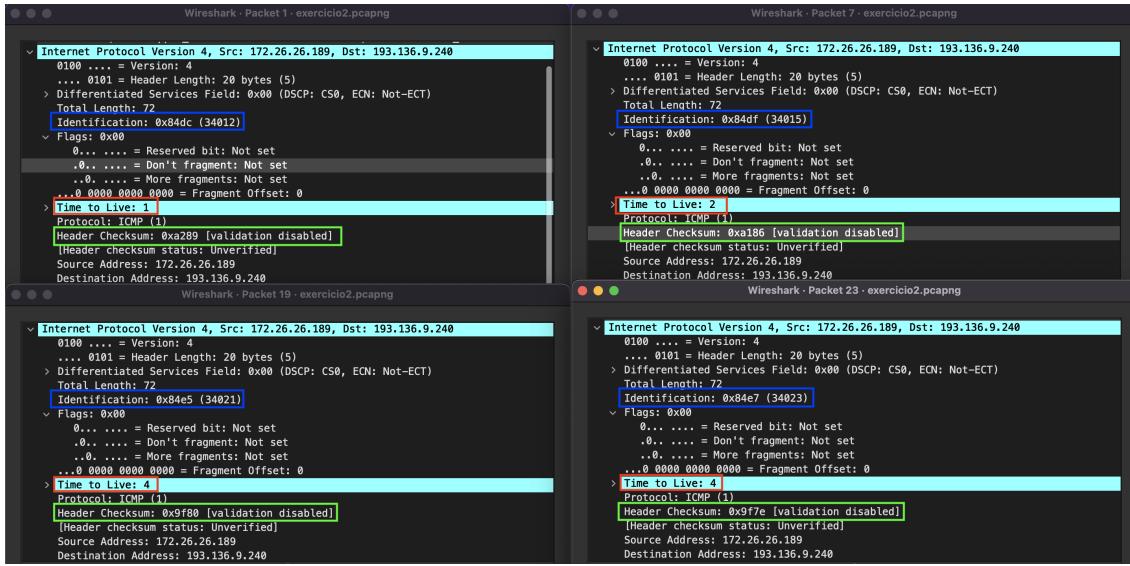


Figura 1.10: 4 Pacotes capturados

1.2.6 Alínea f)

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

Com base na Figura 1.9, conclui-se que os valores do campo de Identificação aumentam de 1 em 1. Já os valores do campo TTL, existem conjuntos de 3 pacotes com TTLs iguais, e futuros conjuntos aumentam o seu TTL por 1 até ser estabelecida ligação com o destinatário.

1.2.7 Alínea 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?

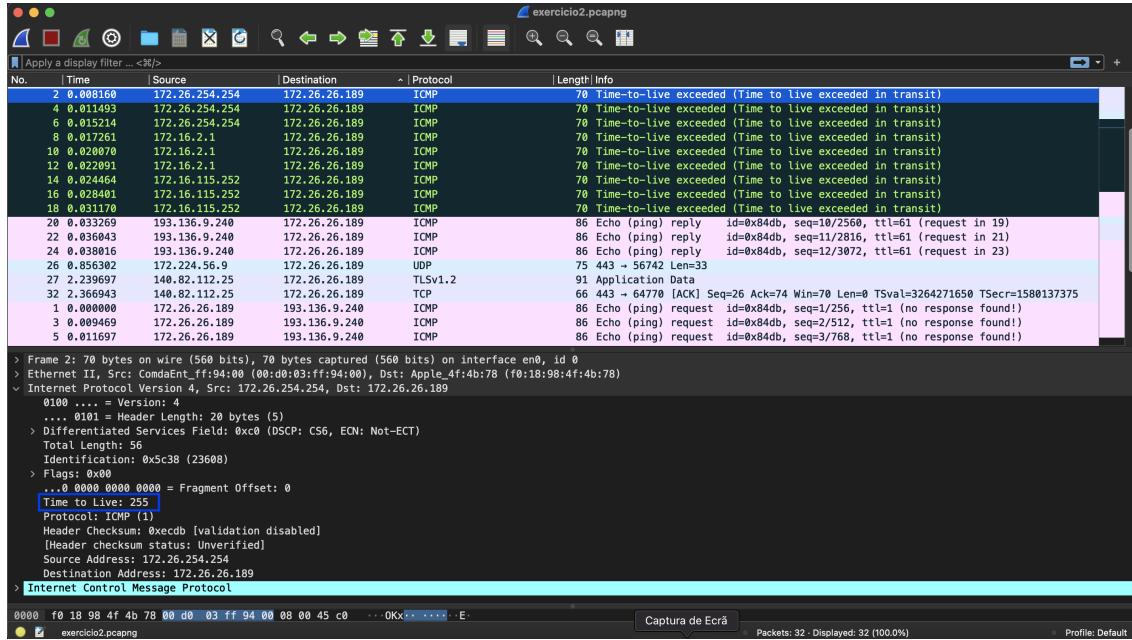


Figura 1.11: Pacotes ordenados por endereço destino

Como podemos ver na Figura 1.11, o valor do campo TTL no primeiro pacote é 255. Este valor decresce 1 de 3 em 3 pacotes, ou seja, os pacotes nº 2, 4 e 6 (pacotes de resposta a pacotes enviados com TTL=1) tem TTL 255, os pacotes nº 8, 10 e 12 (pacotes de resposta a pacotes enviados com TTL=2) tem TTL 254 e os pacotes nº 14, 16 e 18 (pacotes de resposta a pacotes enviados com TTL=3) tem TTL 253.

Isto deve-se ao funcionamento do comando *traceroute*. Os pacotes que são enviados com TTL=1 serão rejeitados ao fim do primeiro passo, sendo enviada uma mensagem de retorno com o valor TTL 255, uma vez que o *router* onde o pacote excedeu o seu TTL, atribuiu o valor 256 à mensagem de retorno e como tem de efectuar 1 salto decrementou o valor dando origem ao valor 255.

Pacotes que sejam rejeitados ao fim do segundo salto retornarão mensagens com TTL=254, visto que essas mensagens terão de executar dois saltos antes de retornarem à interface que enviou o pacote original.

Pacotes que sejam rejeitados ao fim do terceiro salto retornarão mensagens com TTL=253, visto que essas mensagens terão de executar três saltos.

1.3 Exercício 3

Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempo de captura. Observe o tráfego depois do tamanho de pacote ter sido definido para $(4000 + X)$ bytes, em que X representa o número do grupo, sendo neste caso 94.

No.	Time	Source	Destination	Protocol	Length	Info
9	1.0097970	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=b3c2) [Reassembled in #11]
10	1.0098313	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=b3c2) [Reassembled in #11]
11	1.0098282	172.26.26.189	193.136.9.240	ICMP	1148	Echo (ping) request id=0xb3c1, seq=1/256, ttl=1 (no response found!)
12	1.0386067	172.26.254.254	172.26.26.189	ICMP	70	Time-to-Live exceeded (Time to live exceeded in transit)
13	1.0396333	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=b3c3) [Reassembled in #15]
14	1.0397113	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=b3c3) [Reassembled in #15]
15	1.039742	172.26.26.189	193.136.9.240	ICMP	1148	Echo (ping) request id=0xb3c1, seq=2/256, ttl=1 (no response found!)
16	1.042156	172.26.254.254	172.26.26.189	ICMP	70	Time-to-Live exceeded (Time to live exceeded in transit)
17	1.0423993	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=b3c4) [Reassembled in #19]
18	1.042461	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=b3c4) [Reassembled in #19]
19	1.042495	172.26.26.189	193.136.9.240	ICMP	1148	Echo (ping) request id=0xb3c1, seq=3/256, ttl=1 (no response found!)
20	1.045384	172.26.254.254	172.26.26.189	ICMP	70	Time-to-Live exceeded (Time to live exceeded in transit)
21	1.045689	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=b3c5) [Reassembled in #23]
22	1.045732	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=b3c5) [Reassembled in #23]
23	1.045749	172.26.26.189	193.136.9.240	ICMP	1148	Echo (ping) request id=0xb3c1, seq=4/1024, ttl=2 (no response found!)
24	1.048333	172.137.16.75	172.26.26.189	ICMP	70	Time-to-Live exceeded (Time to live exceeded in transit)
25	1.049265	172.26.26.189	193.137.16.65	DNS	83	Standard query 0x6f64 PTR 1.2.16.172.in-addr.arpa
26	1.051211	193.137.16.65	172.26.26.189	DNS	83	Standard query response 0x6f64 Refused PTR 1.2.16.172.in-addr.arpa
27	1.051560	172.26.26.189	193.137.16.145	DNS	83	Standard query 0x6f64 PTR 1.2.16.172.in-addr.arpa
28	1.054233	193.137.16.145	172.26.26.189	DNS	83	Standard query response 0x6f64 Refused PTR 1.2.16.172.in-addr.arpa
29	1.054646	172.26.26.189	193.137.16.75	DNS	83	Standard query 0x6f64 PTR 1.2.16.172.in-addr.arpa
30	1.056707	193.137.16.75	172.26.26.189	DNS	83	Standard query response 0x6f64 Refused PTR 1.2.16.172.in-addr.arpa
31	1.057723	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=b3c6) [Reassembled in #33]
32	1.057836	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=b3c6) [Reassembled in #33]
33	1.057859	172.26.26.189	193.136.9.240	ICMP	1148	Echo (ping) request id=0xb3c1, seq=5/1280, ttl=2 (no response found!)
34	1.068194	172.16.21.1	172.26.26.189	ICMP	70	Time-to-Live exceeded (Time to live exceeded in transit)
35	1.069389	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=b3c7) [Reassembled in #37]
36	1.068465	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=b3c7) [Reassembled in #37]
37	1.068495	172.26.26.189	193.136.9.240	ICMP	1148	Echo (ping) request id=0xb3c1, seq=6/1536, ttl=2 (no response found!)
38	1.062587	172.16.21.1	172.26.26.189	ICMP	70	Time-to-Live exceeded (Time to live exceeded in transit)
39	1.062774	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=b3c8) [Reassembled in #41]
40	1.062824	172.26.26.189	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=b3c8) [Reassembled in #41]
41	1.062841	172.26.26.189	193.136.9.240	ICMP	1148	Echo (ping) request id=0xb3c1, seq=7/1792, ttl=3 (no response found!)
42	1.065678	172.16.115.252	172.26.26.189	ICMP	70	Time-to-Live exceeded (Time to live exceeded in transit)

Figura 1.12: Fragmentação do pacote inicial

1.3.1 Alínea a)

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

Analizando a Figura 1.12, é possível verificar que a primeira mensagem ICMP é a 9. Houve necessidade de fragmentar o pacote inicial, visto que existe um tamanho limite definido pelo protocolo (1514 bytes) é o tamanho do PDU (*Protocol Data Unit*) é 4094 bytes, não sendo possível enviar o pacote inicial sem o fragmentar.

1.3.2 Alínea 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?

Figura 1.13: Primeiro Fragmento do datagrama IP segmento

Observando a Figura 1.13, pode-se afirmar que o datagrama foi fragmentado dado que a flag *More Fragments* está ativa. Pode-se também afirmar que este pacote se trata do primeiro fragmento, baseado na flag *Fragment Offset* que tem valor 0. O tamanho deste datagrama IP fragmentado é 1500 bytes, como é possível verificar no campo *Total Length*.

1.3.3 Alínea 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?

Figura 1.14: 2º fragmento do datagrama IP original

Tomando o mesmo raciocínio da pergunta anterior, e observando a Figura 1.14, o campo *Fragment Offset* indica que este datagrama não se trata do 1º fragmento. Dado que flag *More Fragments* continua ativa, conclui-se que existem mais fragmentos.

1.3.4 Alínea d)

Quantos fragmentos foram criados a partir do datagrama original?

Analizando a captura de *wireshark* e os fragmentos lá existentes, é possível saber que foram criados 3 fragmentos a partir do datagrama original.

1.3.5 Alínea 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.

Os campos que mudam no cabeçalho IP entre os diferentes fragmentos são o *Total Length*, a flag *Fragment Offset* e *More Fragments*, além do *Header Checksum*, campo que não é relevante para a ordenação dos vários fragmentos. Sabendo que os fragmentos têm a mesma identificação e o *offset* de cada um é possível reconstruir o datagrama original de forma fiável.

O primeiro fragmento será aquele cujo valor do campo *Fragment Offset* se encontre a 0. Enquanto a flag *More fragments* se encontre ativa, deve-se continuar a procurar por mais fragmentos, cujo valor do campo *Fragment Offset* será incremental, tendo em conta o tamanho de fragmentos anteriores.

O fragmento final será aquele cujo valor da flag *More Fragments* se encontre a 0.

1.3.6 Alínea f)

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

```

Wireshark · Packet 9 · ejercicio3.pcapng
> Frame 9: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface
> Ethernet II, Src: Apple_4f:4b:78 (f0:18:98:4f:4b:78), Dst: ComdaEnt_ff:94:00 (00:d0)
< Internet Protocol Version 4, Src: 172.26.26.189, 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: 0xb3c2 (46018)
< Flags: 0x20, More fragments
  0... .... = Reserved bit: Not set

Wireshark · Packet 10 · ejercicio3.pcapng
> Frame 10: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface
> Ethernet II, Src: Apple_4f:4b:78 (f0:18:98:4f:4b:78), Dst: ComdaEnt_ff:94:00 (00:d0)
< Internet Protocol Version 4, Src: 172.26.26.189, 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: 0xb3c2 (46018)
< Flags: 0x20, More fragments
  0... .... = Reserved bit: Not set
  .0.. .... = Don't fragment: Not set
  1     - More fragments: Set

Wireshark · Packet 11 · ejercicio3.pcapng
> Frame 11: 1148 bytes on wire (9184 bits), 1148 bytes captured (9184 bits) on interface
> Ethernet II, Src: Apple_4f:4b:78 (f0:18:98:4f:4b:78), Dst: ComdaEnt_ff:94:00 (00:d0)
< Internet Protocol Version 4, Src: 172.26.26.189, 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: 1134
  Identification: 0xb3c2 (46018)
< Flags: 0x01
  0... .... = Reserved bit: Not set
  .0.. .... = Don't fragment: Not set
  ..0. .... = More fragments: Not set

```

Figura 1.15: 3 fragmentos

Tamanho Pacote : Total length - Header Length

Tamanho do pacote: $4094 - 20 = 4074$

Tamanho fragmento 1 : $1500 - 20 = 1480$

Tamanho fragmento 2 : $1500 - 20 = 1480$

Tamanho fragmento 3 : $1134 - 20 = 1114$

Tamanho fragmento 1 + 2 + 3 = Tamanho do Pacote

$1480 + 1480 + 1114 = 4074$

1.3.7 Alínea g)

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

Uma expressão lógica que permitiria detetar esse fragmento seria, por exemplo, a seguinte:

```
# tamanho máximo por fragmento:  
LenMaxFrag = total_Length - header_length  
  
# utilizar divisão inteira:  
n_fragmentos = size_datagrama_original / LenMaxFrag  
# nosso caso: 4094 / 1480 = 2  
  
fragment_offset = n_fragmentos * LenMaxFrag  
# caso: 2960 = 2 * 1480  
# 2960 é o offset do último fragmento  
  
  
if (More_Fragments == 0 && ID == ID_last_fragment  
&& fragment_offset == n_fragmentos * LenMaxFrag )  
    print("This is the last fragment!")  
else:  
    print("This isn't the last fragment!")
```

Quando estamos perante o último fragmento correspondente ao datagrama original, sabemos que a flag *More_Fragments* tem que ser igual a 0, sendo essa umas das condições impostas na expressão lógica, além disso, a identificação do fragmento analisado deve ser igual à dos anteriores.

Capítulo 2

Exercícios - Parte II

2.1 Exercício 1

Atenda aos endereços IP atribuídos automaticamente pelo CORE aos diversos equipamentos da topologia.

2.1.1 Alínea 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.

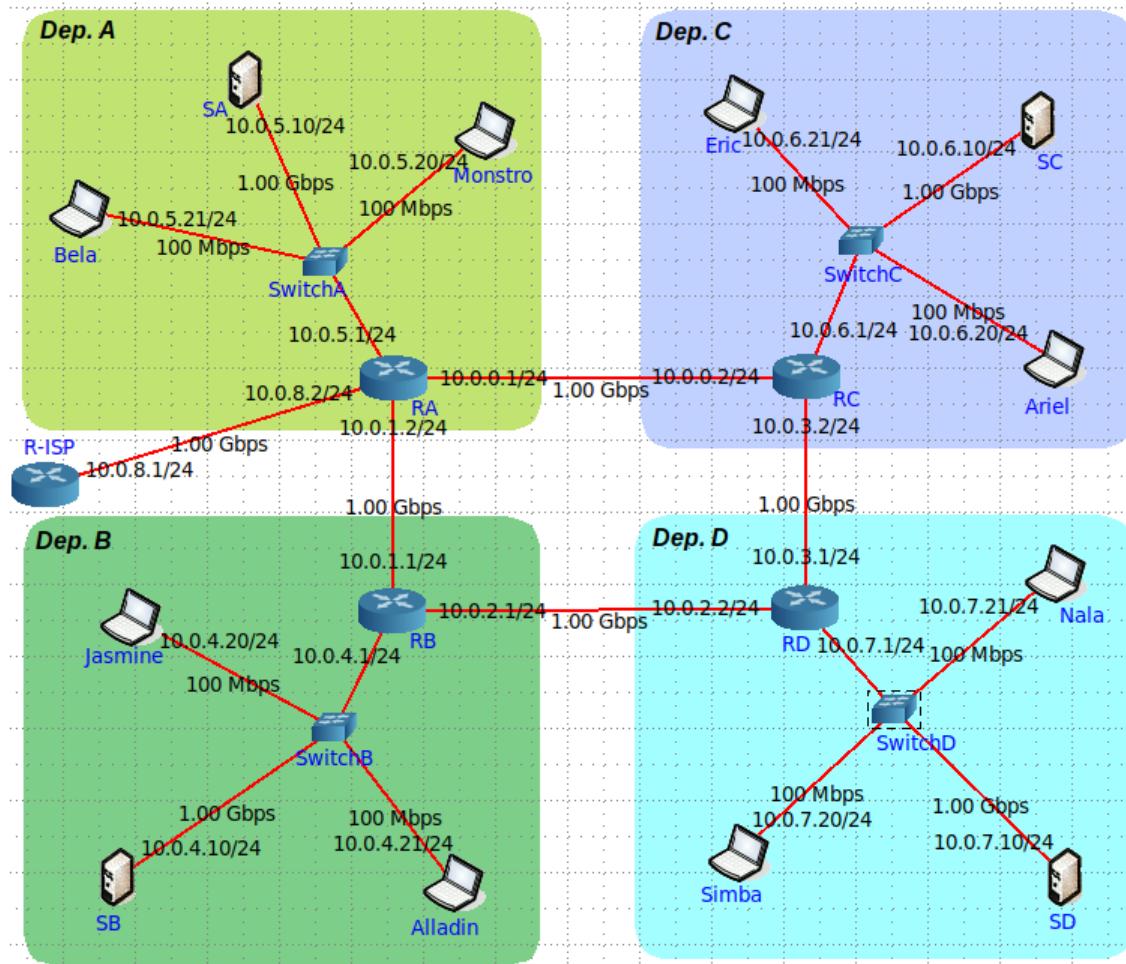


Figura 2.1: Topologia criada

2.1.2 Alínea b)

Tratam-se de endereços públicos ou privados? Porquê?

Os endereços são privados, uma vez que os seus *IPs* se encontram definidos no intervalo entre 10.0.0.0 e 10.255.255.255

2.1.3 Alínea c)

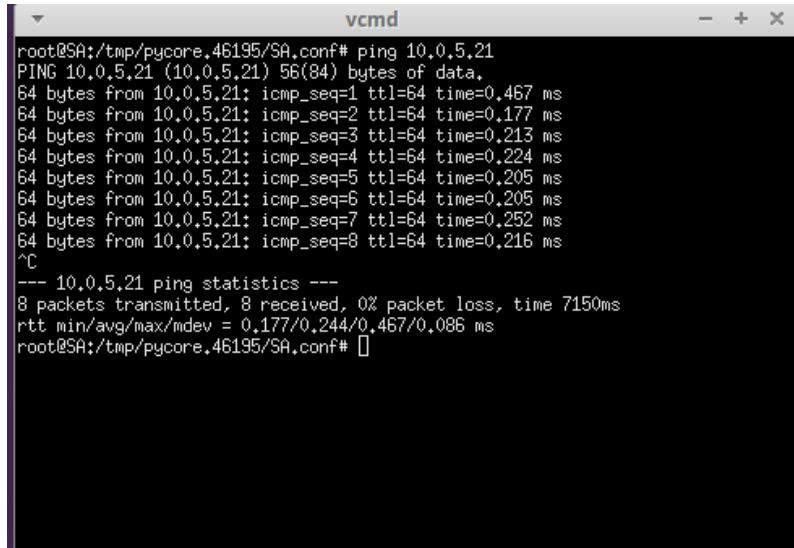
Porque razão não é atribuído um endereço IP aos *switches*?

Um *switch* não funciona ao nível de rede, mas sim ao nível de ligação de dados. Assim, os *switches* não possuem endereços *IPs*, visto que a sua função é apenas transmitir/reencaminhar ligações.

2.1.4 Alínea d)

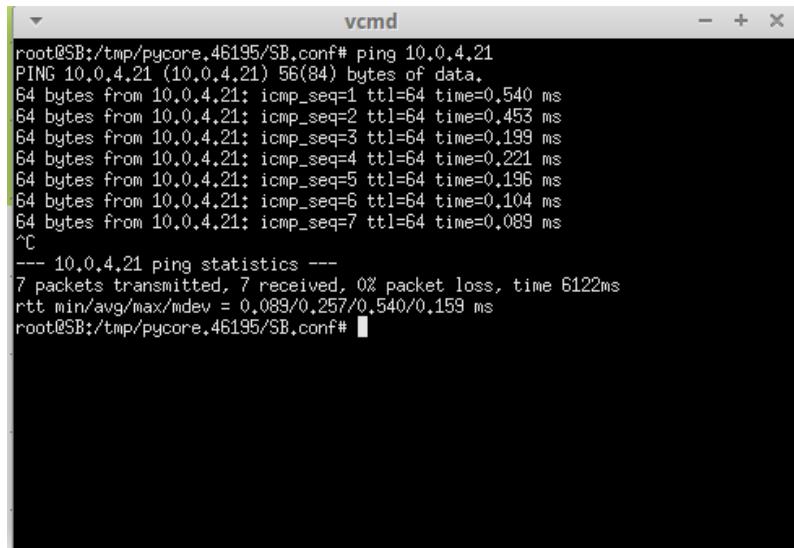
Usando o comando *ping* certifique-se que existe conectividade IP interna a cada departamento (e.g. entre um *laptop* e o servidor respetivo).

Como se verifica nas figuras a seguir, existe conectividade IP interna em cada departamento, uma vez que um *laptop* estabelece ligação com o servidor correspondente do departamento.



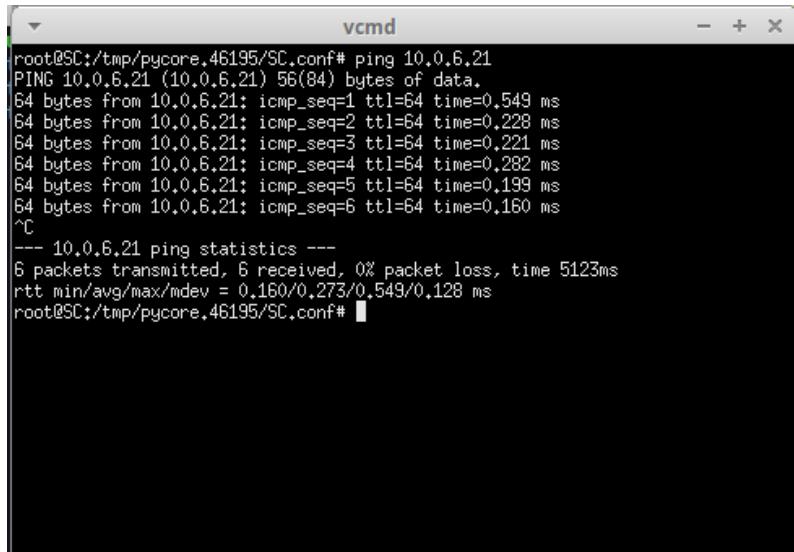
```
root@SA:/tmp/pycore.46195/SA.conf# ping 10.0.5.21
PING 10.0.5.21 (10.0.5.21) 56(84) bytes of data.
64 bytes from 10.0.5.21: icmp_seq=1 ttl=64 time=0.467 ms
64 bytes from 10.0.5.21: icmp_seq=2 ttl=64 time=0.177 ms
64 bytes from 10.0.5.21: icmp_seq=3 ttl=64 time=0.213 ms
64 bytes from 10.0.5.21: icmp_seq=4 ttl=64 time=0.224 ms
64 bytes from 10.0.5.21: icmp_seq=5 ttl=64 time=0.205 ms
64 bytes from 10.0.5.21: icmp_seq=6 ttl=64 time=0.205 ms
64 bytes from 10.0.5.21: icmp_seq=7 ttl=64 time=0.252 ms
64 bytes from 10.0.5.21: icmp_seq=8 ttl=64 time=0.216 ms
^C
--- 10.0.5.21 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7150ms
rtt min/avg/max/mdev = 0.177/0.244/0.467/0.086 ms
root@SA:/tmp/pycore.46195/SA.conf#
```

Figura 2.2: *Ping* entre o servidor SA e o *laptop* Bela



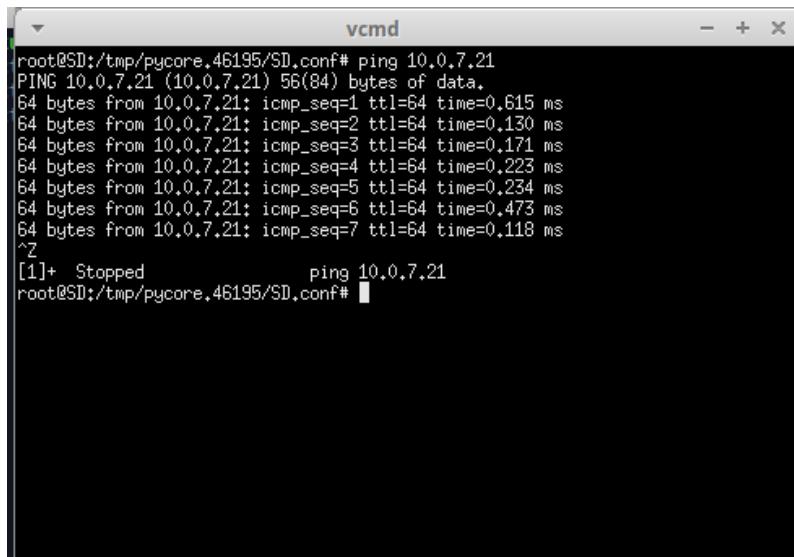
```
root@SB:/tmp/pycore.46195/SB.conf# ping 10.0.4.21
PING 10.0.4.21 (10.0.4.21) 56(84) bytes of data.
64 bytes from 10.0.4.21: icmp_seq=1 ttl=64 time=0.540 ms
64 bytes from 10.0.4.21: icmp_seq=2 ttl=64 time=0.453 ms
64 bytes from 10.0.4.21: icmp_seq=3 ttl=64 time=0.199 ms
64 bytes from 10.0.4.21: icmp_seq=4 ttl=64 time=0.221 ms
64 bytes from 10.0.4.21: icmp_seq=5 ttl=64 time=0.196 ms
64 bytes from 10.0.4.21: icmp_seq=6 ttl=64 time=0.104 ms
64 bytes from 10.0.4.21: icmp_seq=7 ttl=64 time=0.089 ms
^C
--- 10.0.4.21 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6122ms
rtt min/avg/max/mdev = 0.089/0.257/0.540/0.159 ms
root@SB:/tmp/pycore.46195/SB.conf#
```

Figura 2.3: *Ping* entre o servidor SB e o *laptop* Alladin



```
root@SC:/tmp/pycore_46195/SC.conf# ping 10.0.6.21
PING 10.0.6.21 (10.0.6.21) 56(84) bytes of data.
64 bytes from 10.0.6.21: icmp_seq=1 ttl=64 time=0.549 ms
64 bytes from 10.0.6.21: icmp_seq=2 ttl=64 time=0.228 ms
64 bytes from 10.0.6.21: icmp_seq=3 ttl=64 time=0.221 ms
64 bytes from 10.0.6.21: icmp_seq=4 ttl=64 time=0.282 ms
64 bytes from 10.0.6.21: icmp_seq=5 ttl=64 time=0.199 ms
64 bytes from 10.0.6.21: icmp_seq=6 ttl=64 time=0.160 ms
^C
--- 10.0.6.21 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5123ms
rtt min/avg/max/mdev = 0.160/0.273/0.549/0.128 ms
root@SC:/tmp/pycore_46195/SC.conf#
```

Figura 2.4: *Ping* entre o servidor SC e o *laptop* Eric



```
root@SD:/tmp/pycore_46195/SD.conf# ping 10.0.7.21
PING 10.0.7.21 (10.0.7.21) 56(84) bytes of data.
64 bytes from 10.0.7.21: icmp_seq=1 ttl=64 time=0.615 ms
64 bytes from 10.0.7.21: icmp_seq=2 ttl=64 time=0.130 ms
64 bytes from 10.0.7.21: icmp_seq=3 ttl=64 time=0.171 ms
64 bytes from 10.0.7.21: icmp_seq=4 ttl=64 time=0.223 ms
64 bytes from 10.0.7.21: icmp_seq=5 ttl=64 time=0.234 ms
64 bytes from 10.0.7.21: icmp_seq=6 ttl=64 time=0.473 ms
64 bytes from 10.0.7.21: icmp_seq=7 ttl=64 time=0.118 ms
^Z
[1]+  Stopped                  ping 10.0.7.21
root@SD:/tmp/pycore_46195/SD.conf#
```

Figura 2.5: *Ping* entre o servidor SD e o *laptop* Nala

2.1.5 Alínea e)

Execute o número mínimo de comandos *ping* que lhe permite verificar a existência de conectividade IP entre departamentos.

```

root@Bela:/tmp/pycore.46195/Bela.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=62 time=0.500 ms
64 bytes from 10.0.4.20: icmp_seq=2 ttl=62 time=0.312 ms
64 bytes from 10.0.4.20: icmp_seq=3 ttl=62 time=0.315 ms
64 bytes from 10.0.4.20: icmp_seq=4 ttl=62 time=0.107 ms
^C
--- 10.0.4.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3057ms
rtt min/avg/max/mdev = 0.107/0.308/0.500/0.139 ms
root@Bela:/tmp/pycore.46195/Bela.conf#

```

Figura 2.6: Conectividade IP entre o departamento A e o departamento B

```

root@Alladin:/tmp/pycore.46195/Alladin.conf# ping 10.0.7.21
PING 10.0.7.21 (10.0.7.21) 56(84) bytes of data.
64 bytes from 10.0.7.21: icmp_seq=1 ttl=62 time=0.519 ms
64 bytes from 10.0.7.21: icmp_seq=2 ttl=62 time=0.371 ms
64 bytes from 10.0.7.21: icmp_seq=3 ttl=62 time=0.287 ms
64 bytes from 10.0.7.21: icmp_seq=4 ttl=62 time=0.291 ms
^C
--- 10.0.7.21 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4084ms
rtt min/avg/max/mdev = 0.137/0.321/0.519/0.124 ms
ping: write error
root@Alladin:/tmp/pycore.46195/Alladin.conf#

```

Figura 2.7: Conectividade IP entre o departamento B e o departamento D

```

root@Ariel:/tmp/pycore.46195/Ariel.conf# ping 10.0.7.21
PING 10.0.7.21 (10.0.7.21) 56(84) bytes of data.
64 bytes from 10.0.7.21: icmp_seq=1 ttl=62 time=0.520 ms
64 bytes from 10.0.7.21: icmp_seq=2 ttl=62 time=0.291 ms
64 bytes from 10.0.7.21: icmp_seq=3 ttl=62 time=0.134 ms
64 bytes from 10.0.7.21: icmp_seq=4 ttl=62 time=0.115 ms
^C
--- 10.0.7.21 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3058ms
rtt min/avg/max/mdev = 0.115/0.265/0.520/0.162 ms
root@Ariel:/tmp/pycore.46195/Ariel.conf#

```

Figura 2.8: Conectividade IP entre o departamento C e o departamento D

2.1.6 Alínea f)

Verifique se existe conectividade IP do portátil Bela para o *router* de acesso R-ISP

Nesta alínea recorreu-se também, ao comando *ping* para verificar a conectividade entre o portátil Bela e o *router* R-ISP, e como se observa na figura a seguir esta foi estabelecida com sucesso.

```

root@Bela:/tmp/pycore.46195/Bela.conf# ping 10.0.8.1
PING 10.0.8.1 (10.0.8.1) 56(84) bytes of data.
64 bytes from 10.0.8.1: icmp_seq=1 ttl=63 time=0.607 ms
64 bytes from 10.0.8.1: icmp_seq=2 ttl=63 time=0.160 ms
64 bytes from 10.0.8.1: icmp_seq=3 ttl=63 time=0.230 ms
64 bytes from 10.0.8.1: icmp_seq=4 ttl=63 time=0.323 ms
64 bytes from 10.0.8.1: icmp_seq=5 ttl=63 time=0.230 ms
64 bytes from 10.0.8.1: icmp_seq=6 ttl=63 time=0.217 ms
^C
--- 10.0.8.1 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5101ms
rtt min/avg/max/mdev = 0.160/0.294/0.607/0.147 ms
root@Bela:/tmp/pycore.46195/Bela.conf#

```

Figura 2.9: Conectividade entre o portátil Bela e o *router* de acesso R-ISP

2.2 Exercício 2

Para o *router* RA e o portátil Bela:

2.2.1 Alínea 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*)

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
0.0.0.0	10.0.5.1	0.0.0.0	UG	0	0	0	eth0
10.0.5.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0

Figura 2.10: Execução do comando *netstat* no portátil Bela

A primeira linha da tabela de encaminhamento da Figura 2.10 indica o encaminhamento da rota por defeito, que é encaminhado para a interface do *router* RA. A segunda linha refere-se ao encaminhamento de pacotes para uma interface da rede do departamento A.

```

root@RA:/tmp/pycore.35347/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         0.0.0.0       255.255.255.0 U        0 0          0 eth1
10.0.2.0         10.0.1.1      255.255.255.0 UG       0 0          0 eth1
10.0.3.0         10.0.0.2      255.255.255.0 UG       0 0          0 eth0
10.0.4.0         10.0.1.1      255.255.255.0 UG       0 0          0 eth1
10.0.5.0         0.0.0.0       255.255.255.0 U        0 0          0 eth2
10.0.6.0         10.0.0.2      255.255.255.0 UG       0 0          0 eth0
10.0.7.0         10.0.0.2      255.255.255.0 UG       0 0          0 eth0
10.0.8.0         0.0.0.0       255.255.255.0 U        0 0          0 eth3

```

Figura 2.11: Execução do comando *netstat* no *router RA*

No caso da Figura 2.11, as primeiras duas linhas indicam o encaminhamento para uma interface da rede do departamento A. A terceira e quinta linhas indicam o encaminhamento para o endereço do departamento B e para uma interface do *router* desse mesmo departamento, respectivamente. A quarta linha indica o encaminhamento para uma interface da rede do departamento C. A sexta linha indica o encaminhamento para uma interface da rede do departamento A (encaminhamento é direto). A sétima e oitava linhas indicam, respectivamente, o encaminhamento para os endereços do departamento C e D. A última linha indica o encaminhamento para uma interface do *R-ISP*.

2.2.2 Alínea 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).

```

root@Bela:/tmp/pycore.35347/Bela.conf# ps -ax
 PID TTY      STAT   TIME COMMAND
   1 ?        S      0:00 vnoded -v -c /tmp/pycore.35347/Bela -l /tmp/pycore.
  20 pts/2    Ss     0:00 /bin/bash
  28 pts/2    R+     0:00 ps -ax

```

Figura 2.12: Comando *ps -ax* no portátil Bela

Analizando a Figura 2.12 podemos ver que o encaminhamento nos portáteis é estático, visto que não há nenhum processo a correr que indique o contrário, sendo que pode ser alterado manualmente.

```

root@RA:/tmp/pycore.35347/RA.conf# ps -ax
 PID TTY      STAT   TIME COMMAND
   1 ?        S      0:00 vnoded -v -c /tmp/pycore.35347/RA -l /tmp/pycore.35
  75 ?        Ss     0:00 /usr/local/sbin/zebra -d
  81 ?        Ss     0:00 /usr/local/sbin/ospf6d -d
  85 ?        Ss     0:00 /usr/local/sbin/ospfd -d
  93 pts/5    Ss     0:00 /bin/bash
 101 pts/5    R+     0:00 ps -ax

```

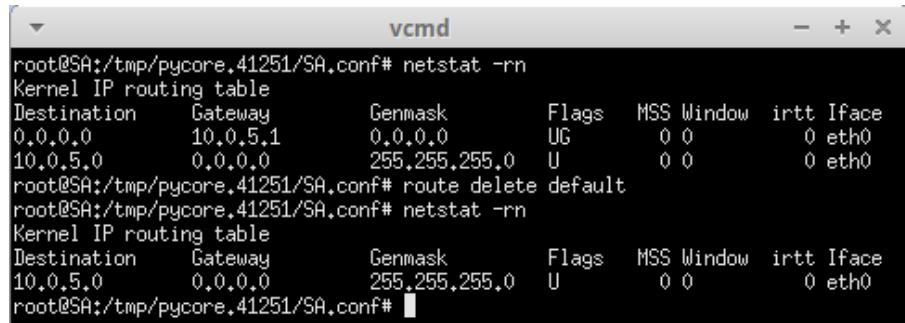
Figura 2.13: Comando *ps -ax* no *router RA*

Está a ser usado encaminhamento dinâmico nos *routers*, visto que o protocolo OSPF6 está a ser executado, que é uma técnica de *dynamic routing*. Este tipo de encaminhamento altera com as alterações na rede.

2.2.3 Alínea 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 SA. Use o comando *route delete* para o efeito. Que implicações tem esta medida para os utilizadores da LEI-RC que acedem ao servidor. Justifique.

Os utilizadores de LEI-RC que não pertençam ao departamento A não vão conseguir aceder ao servidor SA, visto que não possui rota de encaminhamento definida para os IP's que não pertencem à rede desse departamento.

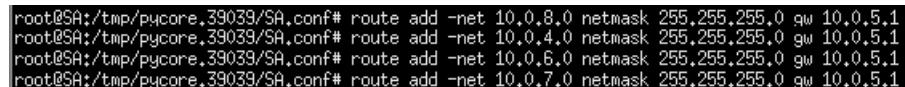


```
root@SA:/tmp/pycore.41251/SA.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         10.0.5.1       0.0.0.0        UG      0 0          0 eth0
10.0.5.0        0.0.0.0        255.255.255.0  U        0 0          0 eth0
root@SA:/tmp/pycore.41251/SA.conf# route delete default
root@SA:/tmp/pycore.41251/SA.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.5.0        0.0.0.0        255.255.255.0  U        0 0          0 eth0
root@SA:/tmp/pycore.41251/SA.conf#
```

Figura 2.14: Comando *route delete*

2.2.4 Alínea d)

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

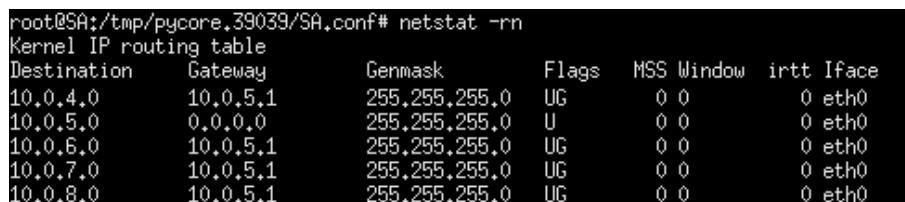


```
root@SA:/tmp/pycore.39039/SA.conf# route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.5.1
root@SA:/tmp/pycore.39039/SA.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.5.1
root@SA:/tmp/pycore.39039/SA.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.5.1
root@SA:/tmp/pycore.39039/SA.conf# route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.5.1
```

Figura 2.15: Comandos *route add*

2.2.5 Alínea 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.39039/SA.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.4.0        10.0.5.1       255.255.255.0  UG      0 0          0 eth0
10.0.5.0        0.0.0.0        255.255.255.0  U        0 0          0 eth0
10.0.6.0        10.0.5.1       255.255.255.0  UG      0 0          0 eth0
10.0.7.0        10.0.5.1       255.255.255.0  UG      0 0          0 eth0
10.0.8.0        10.0.5.1       255.255.255.0  UG      0 0          0 eth0
```

Figura 2.16: Comando *netstat -rn* em SA

```
root@Bela:/tmp/pycore.39039/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=64 time=0.398 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=0.097 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=64 time=0.081 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=64 time=0.077 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=64 time=0.092 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4097ms
rtt min/avg/max/mdev = 0.077/0.149/0.398/0.124 ms
```

Figura 2.17: Comando *ping* no portátil Bela (**Departamento A**)

```
root@Jasmine:/tmp/pycore.39039/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=62 time=0.447 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=0.271 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=0.250 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=0.291 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=0.138 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4082ms
rtt min/avg/max/mdev = 0.138/0.279/0.447/0.099 ms
```

Figura 2.18: Comando *ping* no portátil Jasmine (**Departamento B**)

```
root@Eric:/tmp/pycore.39039/Eric.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=0.336 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=0.157 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=0.145 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=0.141 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=0.139 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4073ms
rtt min/avg/max/mdev = 0.139/0.183/0.336/0.076 ms
```

Figura 2.19: Comando *ping* no portátil Eric (**Departamento C**)

```

root@Simba:/tmp/pycore.39039/Simba.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=61 time=0.651 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=0.195 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=0.601 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=61 time=0.188 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=61 time=0.189 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4100ms
rtt min/avg/max/mdev = 0.188/0.364/0.651/0.213 ms

```

Figura 2.20: Comando *ping* no portátil Simba (**Departamento D**)

2.3 Exercício 3

2.3.1 Alínea 1)

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.

Visto que somos do grupo 94, dispomos do endereço de rede IP 192.168.94.128/25.

Como temos 4 departamentos, e no futuro o número pode vir a aumentar, se utilizássemos só 2 bits não iríamos conseguir acrescentar mais sub-redes, uma vez que com 2 bits temos 4 representações possíveis, logo é melhor optar por 3 bits (8 representações possíveis). Assim dos 7 bits disponíveis para *subnetting*, 3 foram usados para identificação de sub-redes e os restantes 4 para identificação de *host* interfaces.

O endereçamento rege-se-á pelo esquema: **11000000 10101000 01011110 1yyyxxxx**, em que os y's são os bits que identificam a sub-rede e os x's são a identificação do *host* dentro da sub-rede.

Sendo assim atribuiu-se as seguintes sub-redes a cada departamento:

- Departamento A: 192.168.94.128/28
- Departamento B: 192.168.94.144/28
- Departamento C: 192.168.94.160/28
- Departamento D: 192.168.94.176/28

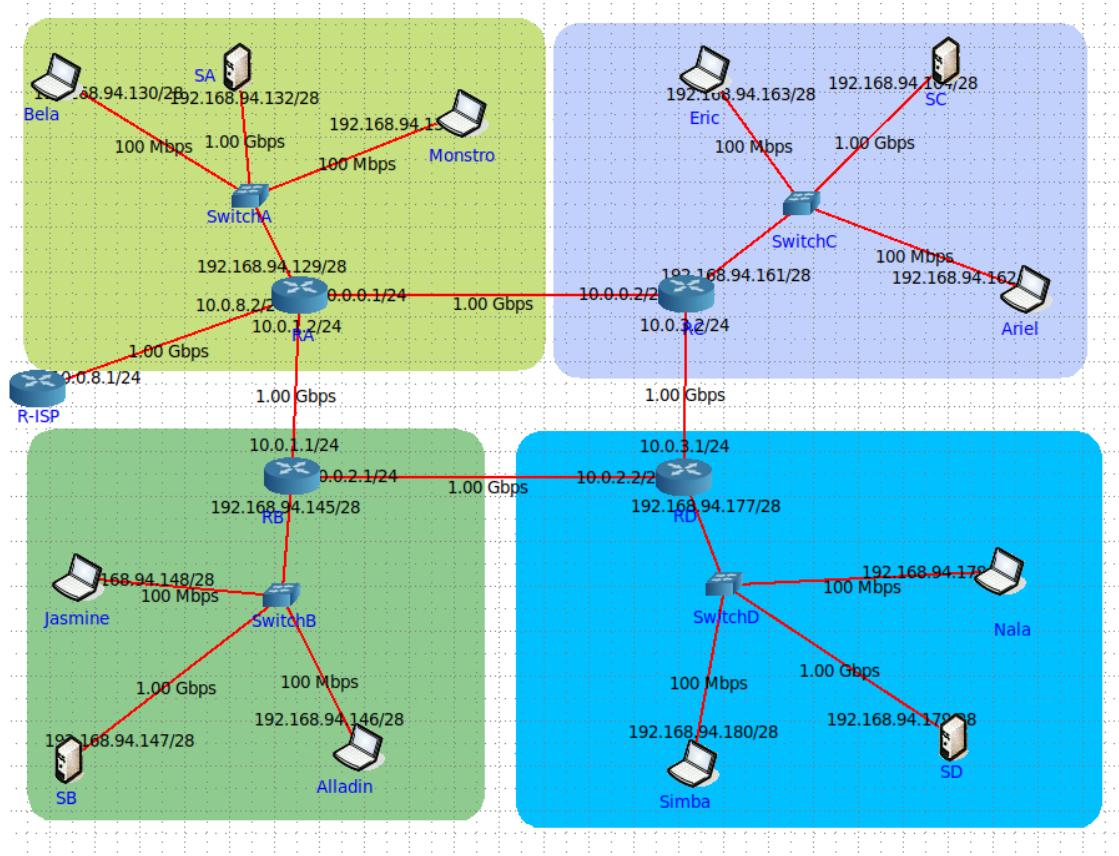


Figura 2.21: Topologia após as alterações feitas

2.3.2 Alínea 2)

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 usada foi a /28 que corresponde a **255.255.255.240**.

Tendo 4 bits para a identificação de *hosts*, considerando que o endereço de rede (caso dos bits todos a 0) e endereço de broadcast (caso dos bits todos a 1) não podem ser usados como endereço de *host*, sobram 14 endereços dos 16 originais ($2^4 - 2$).

Visto que temos 3 bits para identificação de redes, o que possibilita $8 (2^3)$ representações de sub-redes, mas como já temos 4 sub-redes (correspondentes aos 4 departamentos), restam apenas 4 prefixos de sub-redes disponíveis para uso futuro.

2.3.3 Alínea 3)

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.

Para verificarmos a conectividade IP entre as quatro sub-redes, simplesmente executamos duas vezes o comando *ping* num dos *hosts* em cada sub-rede, um para testar a conectividade interna e outro para testar a conectividade entre sub-redes.

```
root@Bela:/tmp/pycore.34953/Bela.conf# ping 192.168.94.132
PING 192.168.94.132 (192.168.94.132) 56(84) bytes of data.
64 bytes from 192.168.94.132: icmp_seq=1 ttl=64 time=0.581 ms
64 bytes from 192.168.94.132: icmp_seq=2 ttl=64 time=0.246 ms
64 bytes from 192.168.94.132: icmp_seq=3 ttl=64 time=0.207 ms
64 bytes from 192.168.94.132: icmp_seq=4 ttl=64 time=0.245 ms
^C
--- 192.168.94.132 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3070ms
rtt min/avg/max/mdev = 0.207/0.319/0.581/0.151 ms
root@Bela:/tmp/pycore.34953/Bela.conf# ping 192.168.94.147
PING 192.168.94.147 (192.168.94.147) 56(84) bytes of data.
64 bytes from 192.168.94.147: icmp_seq=1 ttl=62 time=0.688 ms
64 bytes from 192.168.94.147: icmp_seq=2 ttl=62 time=0.491 ms
64 bytes from 192.168.94.147: icmp_seq=3 ttl=62 time=0.486 ms
^C
--- 192.168.94.147 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 0.486/0.555/0.688/0.094 ms
```

Figura 2.22: Execução do comando *ping* num *host* do departamento A

```
root@Jasmine:/tmp/pycore.34953/Jasmine.conf# ping 192.168.94.147
PING 192.168.94.147 (192.168.94.147) 56(84) bytes of data.
64 bytes from 192.168.94.147: icmp_seq=1 ttl=64 time=0.578 ms
64 bytes from 192.168.94.147: icmp_seq=2 ttl=64 time=0.244 ms
64 bytes from 192.168.94.147: icmp_seq=3 ttl=64 time=0.247 ms
64 bytes from 192.168.94.147: icmp_seq=4 ttl=64 time=0.254 ms
^C
--- 192.168.94.147 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3058ms
rtt min/avg/max/mdev = 0.244/0.330/0.578/0.142 ms
root@Jasmine:/tmp/pycore.34953/Jasmine.conf# ping 192.168.94.179
PING 192.168.94.179 (192.168.94.179) 56(84) bytes of data.
64 bytes from 192.168.94.179: icmp_seq=1 ttl=62 time=0.709 ms
64 bytes from 192.168.94.179: icmp_seq=2 ttl=62 time=0.456 ms
64 bytes from 192.168.94.179: icmp_seq=3 ttl=62 time=0.450 ms
64 bytes from 192.168.94.179: icmp_seq=4 ttl=62 time=0.464 ms
^C
--- 192.168.94.179 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3049ms
rtt min/avg/max/mdev = 0.450/0.519/0.709/0.109 ms
```

Figura 2.23: Execução do comando *ping* num *host* do Departamento B

```

root@Eric:/tmp/pycore.34953/Eric.conf# ping 192.168.94.164
PING 192.168.94.164 (192.168.94.164) 56(84) bytes of data.
64 bytes from 192.168.94.164: icmp_seq=1 ttl=64 time=0.553 ms
64 bytes from 192.168.94.164: icmp_seq=2 ttl=64 time=0.280 ms
64 bytes from 192.168.94.164: icmp_seq=3 ttl=64 time=0.226 ms
64 bytes from 192.168.94.164: icmp_seq=4 ttl=64 time=0.250 ms
^C
--- 192.168.94.164 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3066ms
rtt min/avg/max/mdev = 0.226/0.327/0.553/0.131 ms
root@Eric:/tmp/pycore.34953/Eric.conf# ping 192.168.94.132
PING 192.168.94.132 (192.168.94.132) 56(84) bytes of data.
64 bytes from 192.168.94.132: icmp_seq=1 ttl=62 time=0.725 ms
64 bytes from 192.168.94.132: icmp_seq=2 ttl=62 time=1.32 ms
64 bytes from 192.168.94.132: icmp_seq=3 ttl=62 time=0.471 ms
64 bytes from 192.168.94.132: icmp_seq=4 ttl=62 time=1.31 ms
^C
--- 192.168.94.132 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3057ms
rtt min/avg/max/mdev = 0.471/0.956/1.321/0.369 ms

```

Figura 2.24: Execução do comando *ping* num *host* do departamento C

```

root@Simba:/tmp/pycore.34953/Simba.conf# ping 192.168.94.179
PING 192.168.94.179 (192.168.94.179) 56(84) bytes of data.
64 bytes from 192.168.94.179: icmp_seq=1 ttl=64 time=0.721 ms
64 bytes from 192.168.94.179: icmp_seq=2 ttl=64 time=0.245 ms
64 bytes from 192.168.94.179: icmp_seq=3 ttl=64 time=0.224 ms
64 bytes from 192.168.94.179: icmp_seq=4 ttl=64 time=0.256 ms
^C
--- 192.168.94.179 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.224/0.361/0.721/0.207 ms
root@Simba:/tmp/pycore.34953/Simba.conf# ping 192.168.94.164
PING 192.168.94.164 (192.168.94.164) 56(84) bytes of data.
64 bytes from 192.168.94.164: icmp_seq=1 ttl=62 time=0.701 ms
64 bytes from 192.168.94.164: icmp_seq=2 ttl=62 time=0.443 ms
64 bytes from 192.168.94.164: icmp_seq=3 ttl=62 time=1.37 ms
64 bytes from 192.168.94.164: icmp_seq=4 ttl=62 time=0.493 ms
^X64 bytes from 192.168.94.164: icmp_seq=5 ttl=62 time=1.39 ms
64 bytes from 192.168.94.164: icmp_seq=6 ttl=62 time=0.460 ms
^C
--- 192.168.94.164 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5067ms
rtt min/avg/max/mdev = 0.443/0.810/1.392/0.412 ms

```

Figura 2.25: Execução do comando *ping* num *host* do departamento D

Capítulo 3

Conclusão

Este trabalho permitiu a consolidação dos conceitos relativos a **Datagramas IP e Fragmentação**, bem como de **Endereçamento e Encaminhamento IP**.

Na primeira parte, o objetivo principal foi explorar a constituição dos datagramas IP e a fragmentação dos mesmos, além do funcionamento do comando **traceroute**. Para atingir este objetivo, utilizamos a ferramenta de emulação de redes **CORE** em conjunto com o **Wireshark**, aprofundando os conhecimentos adquiridos anteriormente na UC de Comunicações por Computadores.

No que toca à segunda parte, explorou-se o **Endereçamento e Encaminhamento IP**, através da inspeção e manipulação das tabelas de encaminhamentos dos *routers* presentes na nova topologia. Por fim, a definição manual do endereçamento possibilitou o aprofundamento do conhecimento sobre o *subnetting*.

Concluindo, através deste projeto percebemos a organização indispensável dos vários componentes constituintes de uma rede, bem como a importância de cada um destes para o seu correto funcionamento.