

---

# TP2: Protocolo IPv4: Datagramas IP e Fragmentação

---

TRABALHO REALIZADO POR:

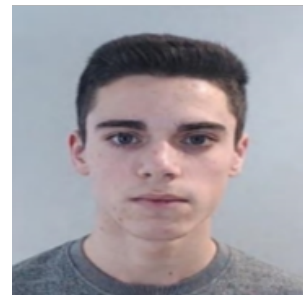
BEATRIZ RIBEIRO MONTEIRO  
PEDRO PEREIRA SOUSA  
TELMO JOSÉ PEREIRA MACIEL



A95437  
Beatriz Monteiro



A95826  
Pedro Sousa



A96569  
Telmo Maciel

# Índice

<b>1</b>	<b>Parte 1</b>	<b>1</b>
1.1	Exercício 1 . . . . .	1
1.2	Exercício 2 . . . . .	3
1.3	Exercício 3 . . . . .	6
<b>2</b>	<b>Parte 2</b>	<b>10</b>
2.1	Exercício 1 . . . . .	11
2.2	Exercício 2 . . . . .	16
2.3	Exercício 3 . . . . .	20
<b>3</b>	<b>Conclusão</b>	<b>23</b>

# 1 Parte 1

## 1.1 Exercício 1

Prepare uma topologia CORE para verificar o comportamento do *traceroute*. Na topologia deve existir: um *host (pc)* cliente designado *Bela* cujo *router* de acesso é R2; o *router* R2 está simultaneamente ligado a dois *routers* R3 e R4; estes estão conectados a um *router* R5, que por sua vez, se liga a um *host (servidor)* designado *Monstro*.

Ajuste o nome dos equipamentos atribuídos por defeito para o enunciado. Nas ligações (*links*) da rede de core estabeleça um tempo de propagação de 10ms.

Após ativar a topologia, note que pode não existir conectividade IP imediata entre a *Bela* e o *Monstro* até que o anúncio de rotas entre *routers* estabilize.

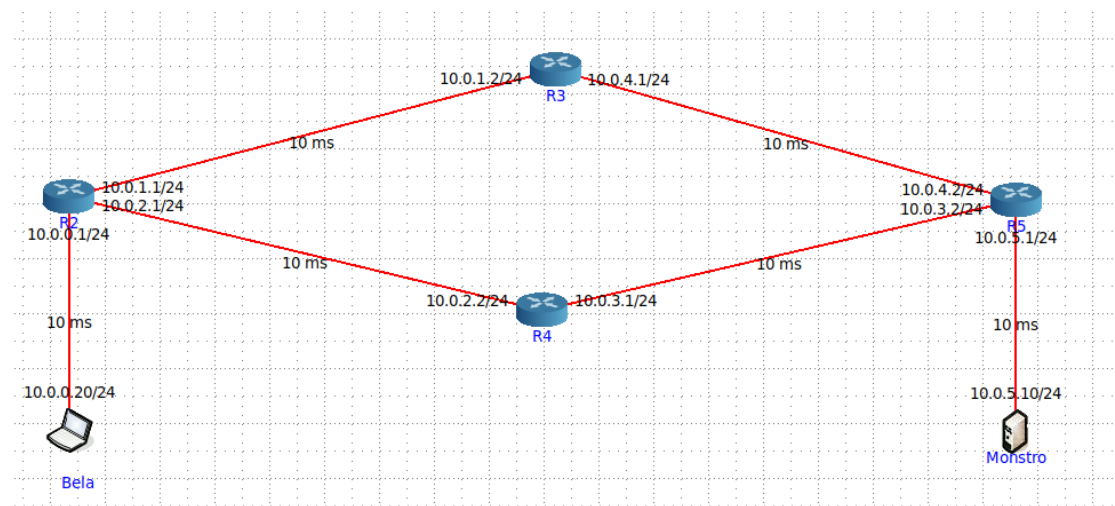


Figure 1: Equipamentos

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

```
root@Bela:/tmp/pycore.42505/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)  41.989 ms  41.949 ms  41.947 ms
 2  10.0.1.2 (10.0.1.2)  63.461 ms  63.462 ms  63.461 ms
 3  10.0.3.2 (10.0.3.2)  83.733 ms  83.735 ms  83.735 ms
 4  10.0.5.10 (10.0.5.10) 125.653 ms 125.652 ms 125.651 ms
root@Bela:/tmp/pycore.42505/Bela.conf#
```

Figure 2: Output do comando *traceroute -I IP do Monstro*

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.

Como podemos verificar pelos resultados, apenas quando o TTL atingiu 4 é que o pacote chegou ao *host*. No entanto, a resposta do *host* ainda demora a chegar, dado que só obtemos a resposta quando TTL atingiu 9. A partir do momento que se obteve

a primeira resposta, deixou-se de mandar pacotes e apenas se recebeu as respostas dos pacotes enviados (a partir da linha 71).

39	34.650776358	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=3/768, ttl=1 (no response found!)
40	34.650771115	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=4/1024, ttl=2 (no response found!)
41	34.650772132	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=5/1280, ttl=2 (no response found!)
42	34.650773156	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=6/1536, ttl=2 (no response found!)
43	34.650773956	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=7/1792, ttl=3 (no response found!)
44	34.650774713	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=8/2048, ttl=3 (no response found!)
45	34.650775602	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=9/2304, ttl=3 (no response found!)
46	34.650776463	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=10/2560, ttl=4 (reply in 71)
47	34.650777254	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=11/2816, ttl=4 (reply in 72)
48	34.650778072	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=12/3072, ttl=4 (reply in 73)
49	34.650778900	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=13/3328, ttl=5 (reply in 74)
50	34.650779727	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=14/3584, ttl=5 (reply in 75)
51	34.650780510	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=15/3840, ttl=5 (reply in 76)
52	34.650781361	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=16/4096, ttl=6 (reply in 77)
53	34.672006322	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
54	34.672015692	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
55	34.672017020	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
56	34.673182753	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=17/4352, ttl=6 (reply in 78)
57	34.673183008	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=18/4608, ttl=6 (reply in 79)
58	34.673187187	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=19/4864, ttl=7 (reply in 80)
59	34.693526232	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
60	34.693535351	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
61	34.693536759	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
62	34.694171037	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=20/5120, ttl=7 (reply in 81)
63	34.694183181	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=21/5376, ttl=7 (reply in 82)
64	34.694183997	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=22/5632, ttl=8 (reply in 83)
65	34.713806598	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
66	34.713814541	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
67	34.713816431	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
68	34.714574913	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=23/5888, ttl=8 (reply in 84)
69	34.714592186	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=24/6144, ttl=8 (reply in 85)
70	34.714601289	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=25/6400, ttl=9 (reply in 86)
71	34.755733854	10.0.0.20	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=10/2560, ttl=61 (request in 46)
72	34.755738021	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=11/2816, ttl=61 (request in 47)
73	34.755738831	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=12/3072, ttl=61 (request in 48)
74	34.755739526	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=13/3328, ttl=61 (request in 49)
75	34.755746224	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=14/3584, ttl=61 (request in 50)
76	34.755746776	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=15/3840, ttl=61 (request in 51)
77	34.755741472	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=16/4096, ttl=61 (request in 52)
78	34.756112145	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=17/4352, ttl=61 (request in 56)
79	34.756130802	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=18/4608, ttl=61 (request in 57)
80	34.756136319	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=19/4864, ttl=61 (request in 58)
81	34.776955626	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=20/5120, ttl=61 (request in 62)
82	34.776959751	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=21/5376, ttl=61 (request in 63)
83	34.776970924	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=22/5632, ttl=61 (request in 64)
84	34.780942520	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=23/5888, ttl=61 (request in 68)
85	34.780946307	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=24/6144, ttl=61 (request in 69)
86	34.780949126	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x0018, seq=25/6400, ttl=61 (request in 70)
87	36.103406892	10.0.0.1	224.0.0.5	OSPF	78 Hello Packet	

Figure 3: Tráfego ICMP

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.

A cada router que passa, o TTL é decrementado. Logo, O TTL tem que ser grande o suficiente para chegar ao Monstro. Se TTL for 4, este chega ao Monstro com valor igual a 1, que é o mínimo possível.

46	34.650776463	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x0018, seq=10/2560, ttl=4 (reply in 71)
----	--------------	-----------	-----------	------	------------------------	---

Figure 4: Valor mínimo do TTL para alcançar o servidor *Monstro*

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.

O valor médio do RTT é 126.652 ms

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, dado que o tempo que demora a ir da origem ao destino, pode não ser (e provavelmente não é) igual ao tempo que demora do destino à origem. O cálculo desta métrica é difícil, porque seria necessário algum recurso contido nos pacotes, de maneira a saber o tempo de chegada a cada *router*.

## 1.2 Exercício 2

```
tracert -I marco.uminho.pt
tracert to marco.uminho.pt (193.136.9.240), 30 hops max, 60 byte packets
 1 _gateway (172.26.254.254)  1.987 ms  1.933 ms  2.715 ms
 2 172.16.2.1 (172.16.2.1)    1.122 ms  1.419 ms  1.879 ms
 3 172.16.115.252 (172.16.115.252)  2.662 ms  3.189 ms  3.176 ms
 4 marco.uminho.pt (193.136.9.240)  2.621 ms  2.611 ms  2.599 ms
```

Figure 5: *Output* do comando `tracert -I marco.uminho.pt`

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

O endereço IP da interface ativa do nosso computador é 172.26.33.116, como é visível na figura 6.

6133 7.771836481	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request id=0x0001, seq=16/4096, ttl=6 (reply in 6152)
6134 7.771998754	172.16.2.1	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)

Figure 6: Endereço IP da Interface Ativa

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

O valor do campo protocolo é 1, como é visível na figura 7, que representa o protocolo ICMP (*Internet Control Message Protocol*).

```
Frame 6118: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 4, Src: 172.26.33.116, 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: 0x61a9 (25001)
  Flags: 0x00
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0xbf11 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.33.116
  Destination Address: 193.136.9.240
Internet Control Message Protocol
```

Figure 7: Valor do Campo Protocolo

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

Atendendo a que o número de *bytes* do campo de dados do datagrama é a diferença entre o tamanho total (*Total Length*) e o tamanho do cabeçalho IPv4 (*Header Length*), que são 60 e 20 *bytes*, respetivamente, podemos concluir que o campo de dados do datagrama tem 40 *bytes*. É possível consultar os valores referidos na Figura 7.

d) O datagrama IP foi fragmentado? Justifique.

Atendendo a que o valor da *Flag* e do *Fragment Offset* se encontra a 0, como é possível ver na figura 7, podemos concluir que o datagrama IP não foi fragmentado.

e) Ordene os pacotes capturados de acordo com o endereço IP fonte (e.g., selecionando o cabeçalho da coluna *Source*), e análise 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.

Como é visível na análise das Figuras 8, 9 e 10, os campos do IPv4 que variam de pacote para pacote é o TTL, a *Identification* e o *Header Checksum*.

6118	7.770819065	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=1/256, ttl=1 (no response found!)
6119	7.770857128	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=2/512, ttl=1 (no response found!)
6120	7.770868024	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=3/768, ttl=1 (no response found!)
6121	7.770880525	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=4/1024, ttl=2 (no response found!)
6122	7.770891420	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=5/1280, ttl=2 (no response found!)
6123	7.770909160	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=6/1536, ttl=2 (no response found!)
6124	7.770920544	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=7/1792, ttl=3 (no response found!)
6125	7.770933744	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=8/2048, ttl=3 (no response found!)
6126	7.770952322	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=9/2304, ttl=3 (no response found!)
6127	7.770964474	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=10/2560, ttl=4 (reply in 6146)
6128	7.770975439	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=11/2816, ttl=4 (reply in 6147)
6129	7.770987382	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=12/3072, ttl=4 (reply in 6148)
6130	7.771001141	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=13/3328, ttl=5 (reply in 6149)
6131	7.771013224	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=14/3584, ttl=5 (reply in 6150)
6132	7.771025027	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=15/3840, ttl=5 (reply in 6151)
6133	7.771036481	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=16/4096, ttl=6 (reply in 6152)
6135	7.772175871	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=17/4352, ttl=6 (reply in 6153)
6137	7.772483313	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=18/4608, ttl=6 (reply in 6156)
6141	7.773377561	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=19/4864, ttl=7 (reply in 6158)
6142	7.773413808	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=20/5120, ttl=7 (reply in 6159)
6143	7.773424773	172.26.33.116	193.136.9.240	ICMP	76 Echo (ping) request	id=0x0001, seq=21/5376, ttl=7 (reply in 6160)

Figure 8: Pacotes ordenados

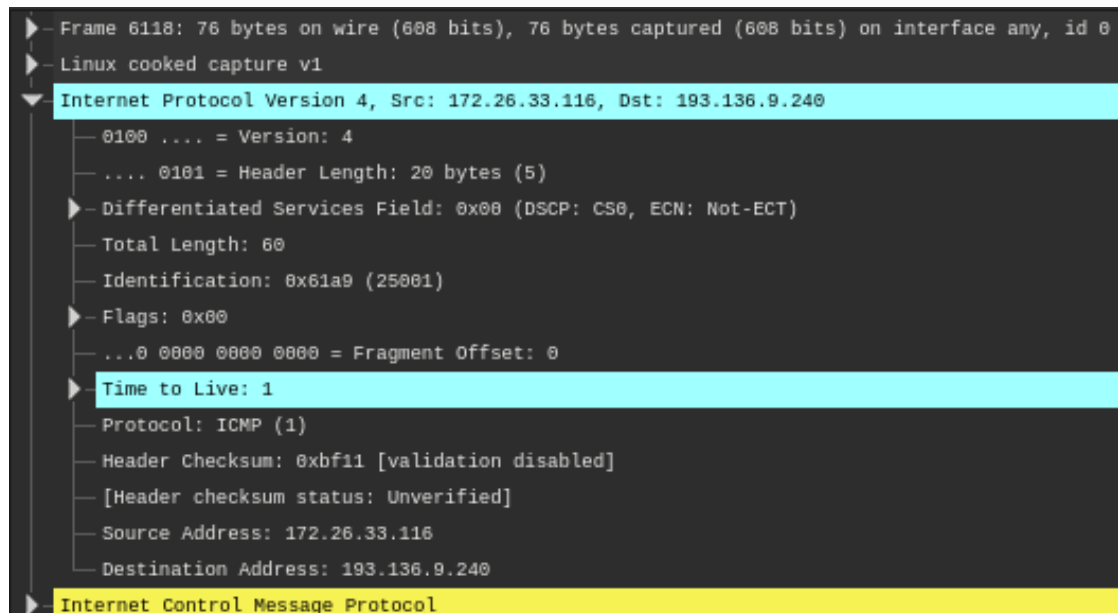


Figure 9: Pacote 1

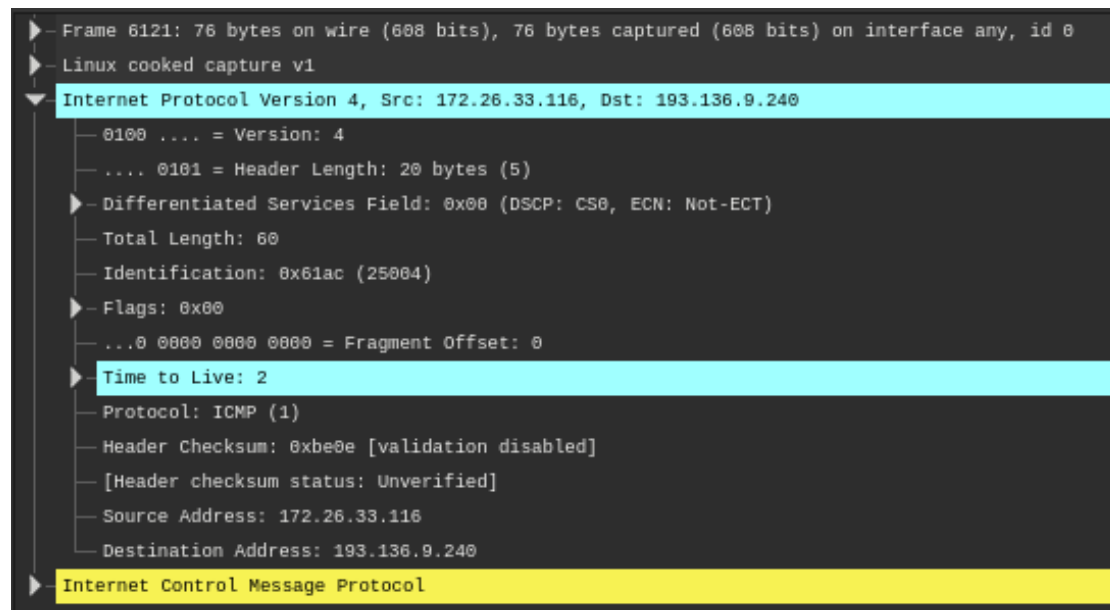


Figure 10: Pacote 3

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

Analisando mais detalhadamente conseguimos concluir que o TTL é incrementado a cada 3 pacotes, a *Identification* é incrementada em 1 e *Header Checksum* é decrementado em 1 a cada pacote.

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?

Para os primeiros 2 pacotes e o 4º, contado apenas aqueles que emitem mensagem de erro, o TTL=253, o 3º, 5º e 6º pacote tem TTL=255 e os últimos três tem TTL=254.

O valor de TTL não permanece constante, dado que para cada valor de ttl inferior a 4 são enviados 3 pacotes (que não chegam ao destino); para cada pacote cujo ttl atingiu 0, o respectivo router envia uma série de pacotes resposta indicando que os pacotes não chegaram ao destino.

6155	7.774119834	172.16.115.252	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6154	7.774118437	172.16.115.252	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6153	7.774117459	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=17/4352, ttl=61 (request in 6135)
6152	7.774116062	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=16/4096, ttl=61 (request in 6133)
6151	7.774115085	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=15/3840, ttl=61 (request in 6132)
6150	7.774112710	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=14/3584, ttl=61 (request in 6131)
6149	7.773584570	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=13/3328, ttl=61 (request in 6130)
6148	7.773583662	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=12/3072, ttl=61 (request in 6129)
6147	7.773582265	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=11/2816, ttl=61 (request in 6128)
6146	7.773581357	193.136.9.240	172.26.33.116	ICMP	76 Echo (ping) reply id=0x0001, seq=10/2560, ttl=61 (request in 6127)
6145	7.773579961	172.26.254.254	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6144	7.773578564	172.16.115.252	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6140	7.772785516	172.26.254.254	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6139	7.772784049	172.26.254.254	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6138	7.772783141	172.16.2.1	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6136	7.772396265	172.16.2.1	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)
6134	7.771998754	172.16.2.1	172.26.33.116	ICMP	72 Time-to-live exceeded (Time to live exceeded in transit)

Figure 11: Pacote 3



### 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 4040 *bytes*.

```
tracert -I marco.uminho.pt 4040
tracert to marco.uminho.pt (193.136.9.240), 30 hops max, 4040 byte packets
 1 _gateway (172.26.254.254)  2.891 ms  4.768 ms  5.053 ms
 2 172.16.2.1 (172.16.2.1)    4.413 ms  4.951 ms  4.914 ms
 3 172.16.115.252 (172.16.115.252)  4.897 ms  4.862 ms  6.026 ms
 4 marco.uminho.pt (193.136.9.240)  4.769 ms  5.967 ms  5.934 ms
```

Figure 12: Terminal após o *tracert*

No.	Time	Source	Destination	Protocol	Length Info
10	4.189584750	172.26.120.162	193.137.16.65	DNS	75 Standard query 0xec1b A marco.uminho.pt
11	4.189628540	172.26.120.162	193.137.16.65	DNS	75 Standard query 0x8b0e AAAA marco.uminho.pt
12	4.194487699	193.137.16.65	172.26.120.162	DNS	129 Standard query response 0x8b0e AAAA marco.uminho.pt SOA dns.uminho.pt
13	4.196065585	193.137.16.65	172.26.120.162	DNS	347 Standard query response 0xec1b A marco.uminho.pt A 193.136.9.240 NS dns3.uminho.pt
14	4.196452567	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fadc) [Reassembled in #16]
15	4.196498242	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fadc) [Reassembled in #16]
16	4.196508718	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no response found!)
17	4.196534907	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fadd) [Reassembled in #19]
18	4.196544545	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fadd) [Reassembled in #19]
19	4.196554113	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=2/512, ttl=1 (no response found!)
20	4.196574576	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fade) [Reassembled in #22]
21	4.196594633	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fade) [Reassembled in #22]
22	4.196593614	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=3/768, ttl=1 (no response found!)
23	4.196612089	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fadf) [Reassembled in #25]
24	4.196621997	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fadf) [Reassembled in #25]
25	4.196631356	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=4/1024, ttl=2 (no response found!)
26	4.196648676	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae0) [Reassembled in #28]
27	4.196658105	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae0) [Reassembled in #28]
28	4.196669999	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=5/1280, ttl=2 (no response found!)
29	4.196689113	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae1) [Reassembled in #31]
30	4.196700986	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae1) [Reassembled in #31]
31	4.196710135	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=6/1536, ttl=2 (no response found!)
32	4.196729131	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae2) [Reassembled in #34]
33	4.196738280	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae2) [Reassembled in #34]
34	4.196747299	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=7/1792, ttl=3 (no response found!)
35	4.196756378	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae3) [Reassembled in #37]
36	4.196775595	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae3) [Reassembled in #37]
37	4.196785352	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=8/2048, ttl=3 (no response found!)
38	4.196803161	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae4) [Reassembled in #40]
39	4.196812171	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae4) [Reassembled in #40]
40	4.196821116	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=9/2304, ttl=3 (no response found!)
41	4.196830919	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae5) [Reassembled in #43]
42	4.196840278	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae5) [Reassembled in #43]
43	4.196850938	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=10/2560, ttl=4 (reply in 76)
44	4.196876563	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae6) [Reassembled in #46]
45	4.196885921	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae6) [Reassembled in #46]
46	4.196894442	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=11/2816, ttl=4 (reply in 86)
47	4.196912111	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae7) [Reassembled in #49]
48	4.196921699	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae7) [Reassembled in #49]
49	4.196930269	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=12/3072, ttl=4 (reply in 89)
50	4.196949755	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fae8) [Reassembled in #52]
51	4.196958834	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fae8) [Reassembled in #52]
52	4.196967983	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=13/3328, ttl=5 (reply in 92)

Figure 13: *Output do traceroute*

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

A MTU que estamos a usar só é capaz de enviar pacotes com tamanho de 1500 *bytes*. Como o nosso pacote possui 4040 *bytes*, existe a necessidade de o fragmentar para que possa ser enviado.

14	4.196452567	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=fadc) [Reassembled in #16]
15	4.196498242	172.26.120.162	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=fadc) [Reassembled in #16]
16	4.196508718	172.26.120.162	193.136.9.240	ICMP	1094 Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no response found!)

Figure 14: Fragmentação do primeiro pacote

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?

A informação contida em *Flags* permite-nos saber que o datagrama foi fragmentado uma vez que o '*Fragment Offset*' se encontra com o valor 0 e o '*More fragments*' com o



valor 1. Estes valores indicam-nos também que se trata do primeiro fragmento. Quanto ao tamanho do datagrama sabemos que é de 1500 *bytes*.

```
▶ Frame 14: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlo1, id 0
▶ Ethernet II, Src: IntelCor_38:aa:54 (e0:d4:e8:38:aa:54), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.120.162, 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: 0xfadc (64220)
    ▼ 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: 0xa90f [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 172.26.120.162
        Destination Address: 193.136.9.240
        [Reassembled IPv4 in frame: 16]
    ▼ Data (1480 bytes)
        Data: 0800c6fb0004000148494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f60616263...
        [Length: 1480]
```

Figure 15: *Flags* do primeiro segmento

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?

Como foi dito na questão anterior, sabemos que se trata do primeiro fragmento quando o valor da *flag* 'Fragment Offset' é 0 e o valor de 'More fragments' é 1. Neste caso, o valor da *flag* 'Fragment Offset' é diferente de zero, o que nos leva a concluir que não é o primeiro fragmento. Uma vez que o valor da *flag* 'More fragments' continua a ter o valor 1 podemos concluir que ainda existem mais fragmentos.

```
▶ Frame 15: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlo1, id 0
▶ Ethernet II, Src: IntelCor_38:aa:54 (e0:d4:e8:38:aa:54), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.120.162, 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: 0xfadc (64220)
    ▼ 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: 0xa856 [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 172.26.120.162
        Destination Address: 193.136.9.240
        [Reassembled IPv4 in frame: 16]
    ▼ Data (1480 bytes)
        Data: 48494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f606162636465666768696a6b...
        [Length: 1480]
```

Figure 16: *Flags* do segundo fragmento

d) Quantos fragmentos foram criados a partir do datagrama original?

Sabemos que estamos na presença do último fragmento quando a *flag* 'More fragments' se encontra a 0 (o que podemos ver na figura 17). Posto isto, como sabemos que este fragmento é o último, podemos contar o número total de fragmentos, ou seja, 3.

```

> Frame 16: 1094 bytes on wire (8752 bits), 1094 bytes captured (8752 bits) on interface wlo1, id 0
> Ethernet II, Src: IntelCor_38:aa:54 (e0:d4:e8:38:aa:54), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
- Internet Protocol Version 4, Src: 172.26.120.162, 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: 1080
  Identification: 0xfadc (64220)
  > 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: 0xc941 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.120.162
  Destination Address: 193.136.9.240
  > [3 IPv4 Fragments (4020 bytes): #14(1480), #15(1480), #16(1060)]
  > Internet Control Message Protocol

```

Figure 17: *Flags* do terceiro fragmento

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 os valores que mudam no cabeçalho IP são os valores correspondentes ao '*Fragment Offset*' e '*More fragments*'. A *flag* '*Fragment Offset*' permite-nos saber qual a ordem que devemos organizar os fragmentos uma vez que estes aparecem por ordem crescente e a *flag* '*More fragments*' permite-nos averiguar se existem mais fragmentos do datagrama original. Juntando estas duas *flags* podemos reconstruir o datagrama original.

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

Sabendo que a MTU que estamos a usar só é capaz de enviar pacotes com o tamanho máximo de 1500 *bytes* e que 20 *bytes* fazem parte do cabeçalho, concluímos que só serão enviados 1480 *bytes* de dados por fragmento.

Como no nosso caso é necessário enviar 4040 *bytes* de dados, vamos precisar de dividir a nossa mensagem em 3 fragmentos com 1480, 1480 e 1080 *bytes*, respetivamente. Podemos verificar isto tudo na seguinte tabela:

Fragmento	<i>Bytes</i> totais	<i>Bytes</i> do cabeçalho	<i>Bytes</i> de dados	<i>More fragments</i>	<i>Fragment Offset</i>
1	1500	20	1480	1	0
2	1500	20	1480	1	1480
3	1100	20	1080	0	2960

Table 1: Fragmentação

---

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

Para detetar o último fragmento do datagrama original basta-nos verificar as condições "More fragments"=0 e "Fragment Offset"!=0.

## 2 Parte 2

Considere que a topologia de rede LEI-RC é distribuída por quatro departamentos (A, B, C e D) e cada departamento possui um router de acesso à sua rede local. Estes routers de acesso (RA, RB, RC e RD) estão interligados entre si por ligações Ethernet a 1Gbps, formando um anel. Por sua vez, existe um servidor por departamento (SA, SB, SC, SD) e dois portáteis (*pc*) por departamento (A - Bela, Monstro; B - Jasmine, Alladin; C - Ariel, Eric; D - Simba e Nala), todos interligados ao router respectivo através de um comutador (*switch*). Cada servidor S tem uma ligação a 1Gbps e os *laptops* ligações a 100Mbps. Considere apenas a existência de um comutador por departamento.

A conectividade IP externa da organização é assegurada através de um router de acesso RISP conectado a RA por uma ligação ponto-a-ponto a 1 Gbps. Construa uma topologia CORE que reflita a rede local da organização. Atribua as designações corretas aos equipamentos. Para facilitar a visualização pode ocultar o endereçamento IPv6. Grave a topologia para eventual reposição futura.

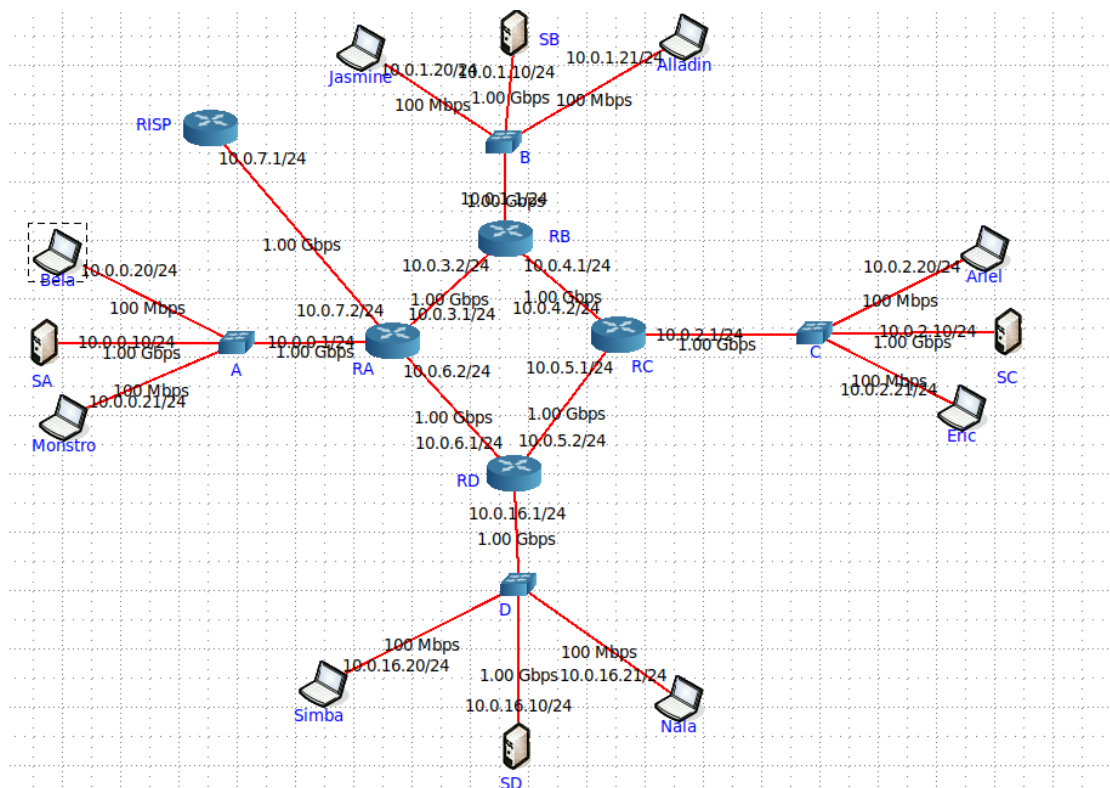


Figure 18: Equipamentos e Departamentos

---

## 2.1 Exercício 1

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

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.

Os equipamentos têm a máscara /24, o que corresponde a 255.255.255.0. Os endereços IP atribuídos a cada equipamento podem ser observados na figura 18.

b) Tratam-se de endereços públicos ou privados? Porquê?

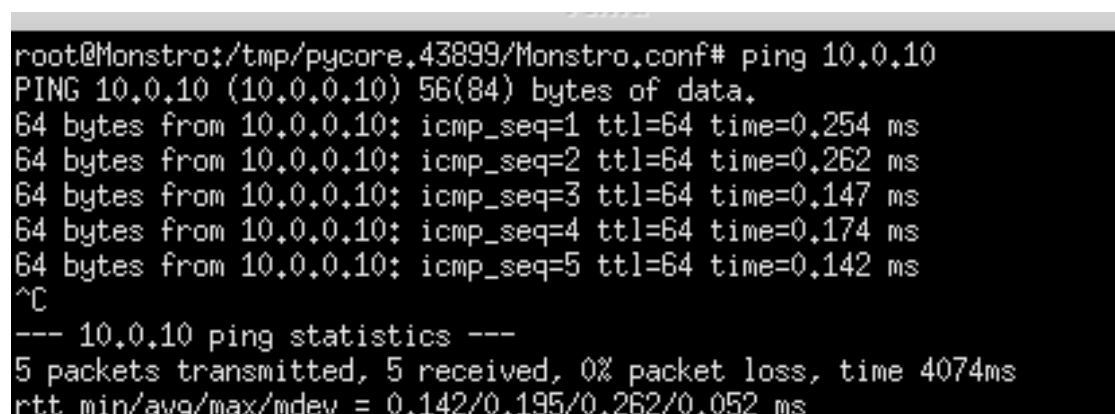
Todos os endereços pertencem ao bloco 10.0.0.0 - 10.255.255.255 são endereços privados. Como os endereços IPs de todos os equipamentos começam por 10, podemos concluir que se tratam de endereços privados.

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

Dado a principal função dos switches ser reencaminhamento de informação apenas registando os endereços MAC dos vários dispositivos ligados a si, não lhe é atribuído um endereço IP porque não há necessidade, visto que apenas reencaminham pacotes para o destino.

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

Tendo em conta que o *packet loss* é de 0%, podemos concluir que existe conectividade IP interna a cada departamento.



```
root@Monstro:/tmp/pycore.43899/Monstro.conf# ping 10.0.10
PING 10.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=0.254 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=0.262 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=0.147 ms
64 bytes from 10.0.0.10: icmp_seq=4 ttl=64 time=0.174 ms
64 bytes from 10.0.0.10: icmp_seq=5 ttl=64 time=0.142 ms
^C
--- 10.0.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4074ms
rtt min/avg/max/mdev = 0.142/0.195/0.262/0.052 ms
```

Figure 19: *Output* do comando *ping* no Departamento A

```
root@Alladin:/tmp/pycore.43899/Alladin.conf# ping 10.0.1.10
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
64 bytes from 10.0.1.10: icmp_seq=1 ttl=64 time=0.429 ms
64 bytes from 10.0.1.10: icmp_seq=2 ttl=64 time=0.076 ms
64 bytes from 10.0.1.10: icmp_seq=3 ttl=64 time=0.138 ms
64 bytes from 10.0.1.10: icmp_seq=4 ttl=64 time=0.122 ms
^C
--- 10.0.1.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
rtt min/avg/max/mdev = 0.076/0.191/0.429/0.139 ms
root@Alladin:/tmp/pycore.43899/Alladin.conf#
```

Figure 20: *Output do comando ping no Departamento B*

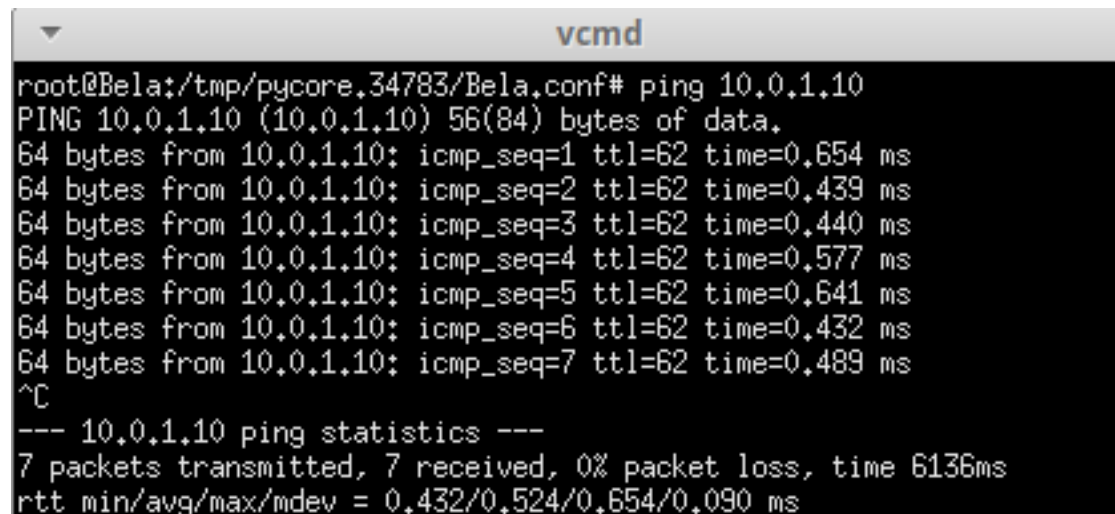
```
root@Eric:/tmp/pycore.43899/Eric.conf# ping 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=64 time=0.249 ms
64 bytes from 10.0.2.10: icmp_seq=2 ttl=64 time=0.173 ms
64 bytes from 10.0.2.10: icmp_seq=3 ttl=64 time=0.170 ms
64 bytes from 10.0.2.10: icmp_seq=4 ttl=64 time=0.131 ms
64 bytes from 10.0.2.10: icmp_seq=5 ttl=64 time=0.148 ms
^C
--- 10.0.2.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4094ms
rtt min/avg/max/mdev = 0.131/0.174/0.249/0.040 ms
root@Eric:/tmp/pycore.43899/Eric.conf#
```

Figure 21: *Output do comando ping no Departamento C*

```
root@Simba:/tmp/pycore.43899/Simba.conf# ping 10.0.16.10
PING 10.0.16.10 (10.0.16.10) 56(84) bytes of data.
64 bytes from 10.0.16.10: icmp_seq=1 ttl=64 time=0.404 ms
64 bytes from 10.0.16.10: icmp_seq=2 ttl=64 time=0.200 ms
64 bytes from 10.0.16.10: icmp_seq=3 ttl=64 time=0.148 ms
64 bytes from 10.0.16.10: icmp_seq=4 ttl=64 time=0.586 ms
64 bytes from 10.0.16.10: icmp_seq=5 ttl=64 time=0.136 ms
^C
--- 10.0.16.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4080ms
rtt min/avg/max/mdev = 0.136/0.294/0.586/0.174 ms
root@Simba:/tmp/pycore.43899/Simba.conf# S
```

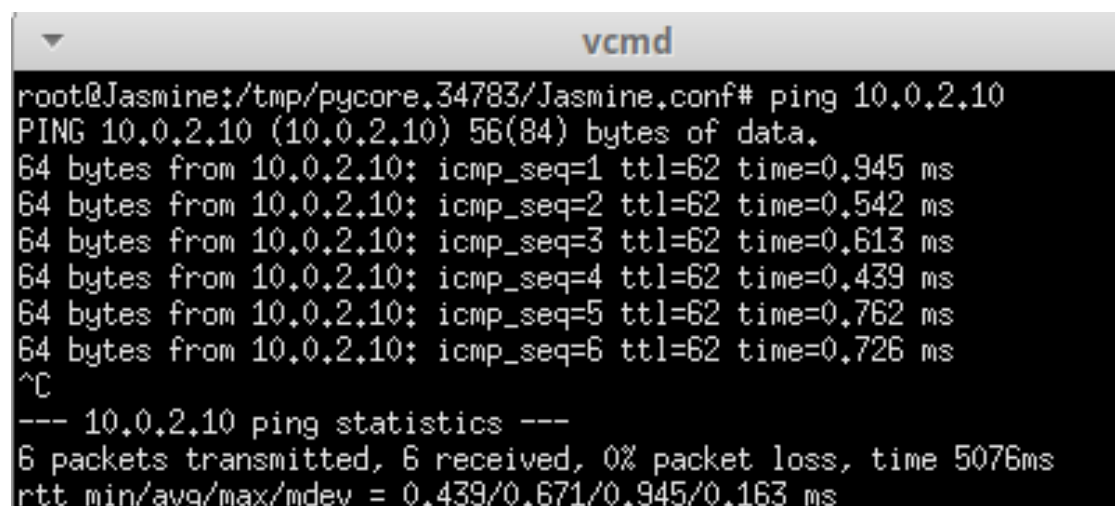
Figure 22: *Output do comando ping no Departamento D*

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



```
vcmd
root@Bela:/tmp/pycore.34783/Bela.conf# ping 10.0.1.10
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data.
64 bytes from 10.0.1.10: icmp_seq=1 ttl=62 time=0.654 ms
64 bytes from 10.0.1.10: icmp_seq=2 ttl=62 time=0.439 ms
64 bytes from 10.0.1.10: icmp_seq=3 ttl=62 time=0.440 ms
64 bytes from 10.0.1.10: icmp_seq=4 ttl=62 time=0.577 ms
64 bytes from 10.0.1.10: icmp_seq=5 ttl=62 time=0.641 ms
64 bytes from 10.0.1.10: icmp_seq=6 ttl=62 time=0.432 ms
64 bytes from 10.0.1.10: icmp_seq=7 ttl=62 time=0.489 ms
^C
--- 10.0.1.10 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6136ms
rtt min/avg/max/mdev = 0.432/0.524/0.654/0.090 ms
```

Figure 23: Verificar conectividade entre o *Laptop* Bela (Departamento A) e o Departamento B



```
vcmd
root@Jasmine:/tmp/pycore.34783/Jasmine.conf# ping 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_seq=1 ttl=62 time=0.945 ms
64 bytes from 10.0.2.10: icmp_seq=2 ttl=62 time=0.542 ms
64 bytes from 10.0.2.10: icmp_seq=3 ttl=62 time=0.613 ms
64 bytes from 10.0.2.10: icmp_seq=4 ttl=62 time=0.439 ms
64 bytes from 10.0.2.10: icmp_seq=5 ttl=62 time=0.762 ms
64 bytes from 10.0.2.10: icmp_seq=6 ttl=62 time=0.726 ms
^C
--- 10.0.2.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5076ms
rtt min/avg/max/mdev = 0.439/0.671/0.945/0.163 ms
```

Figure 24: Verificar conectividade entre o *Laptop* Jasmine (Departamento B) e o Departamento C



```
vcmd
root@Ariel:/tmp/pycore.34783/Ariel.conf# ping 10.0.16.10
PING 10.0.16.10 (10.0.16.10) 56(84) bytes of data.
64 bytes from 10.0.16.10: icmp_seq=1 ttl=62 time=1.28 ms
64 bytes from 10.0.16.10: icmp_seq=2 ttl=62 time=0.415 ms
64 bytes from 10.0.16.10: icmp_seq=3 ttl=62 time=0.657 ms
64 bytes from 10.0.16.10: icmp_seq=4 ttl=62 time=1.69 ms
64 bytes from 10.0.16.10: icmp_seq=5 ttl=62 time=0.630 ms
64 bytes from 10.0.16.10: icmp_seq=6 ttl=62 time=1.21 ms
^C
--- 10.0.16.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5068ms
rtt min/avg/max/mdev = 0.415/0.979/1.685/0.444 ms
```

Figure 25: Verificar conectividade entre o *Laptop* Ariel (Departamento C) e o Departamento D

```
root@Nala:/tmp/pycore.38987/Nala.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=62 time=0.578 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=62 time=0.269 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=62 time=0.294 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2030ms
rtt min/avg/max/mdev = 0.269/0.380/0.578/0.140 ms
root@Nala:/tmp/pycore.38987/Nala.conf# █
```

Figure 26: Verificar conectividade entre o *Laptop* Nala (Departamento D) e o Departamento A

Para verificar a existência de conectividade entre departamentos basta-nos dar ping do portátil do nosso departamento para um único host do departamento a testar a conectividade pois, através da pergunta anterior, sabemos que existe também conectividade interna a cada departamento. Posto isto, como sabemos que o comando ping envia e recebe dados, provamos que há conectividade.

---

f) Verifique se existe conectividade IP do portátil Bela para o router de acesso RISP.

```
root@Bela:/tmp/pycore.43899/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=0.232 ms
64 bytes from 10.0.7.1: icmp_seq=2 ttl=63 time=0.227 ms
64 bytes from 10.0.7.1: icmp_seq=3 ttl=63 time=0.222 ms
64 bytes from 10.0.7.1: icmp_seq=4 ttl=63 time=0.117 ms
64 bytes from 10.0.7.1: icmp_seq=5 ttl=63 time=0.269 ms
64 bytes from 10.0.7.1: icmp_seq=6 ttl=63 time=0.203 ms
64 bytes from 10.0.7.1: icmp_seq=7 ttl=63 time=0.128 ms
^C
--- 10.0.7.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6142ms
rtt min/avg/max/mdev = 0.117/0.199/0.269/0.052 ms
root@Bela:/tmp/pycore.43899/Bela.conf# █
```

Figure 27: Verificação da conectividade pelo *ping*

---

## 2.2 Exercício 2

Para o router RA e o portátil Bela:

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*).

```
root@Bela:/tmp/pycore.34783/Bela.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt Iface
0.0.0.0          10.0.0.1         0.0.0.0         UG      0 0        0 eth0
10.0.0.0         0.0.0.0          255.255.255.0   U       0 0        0 eth0
root@Bela:/tmp/pycore.34783/Bela.conf#
```

Figure 28: Tabela de encaminhamento do *laptop* Bela

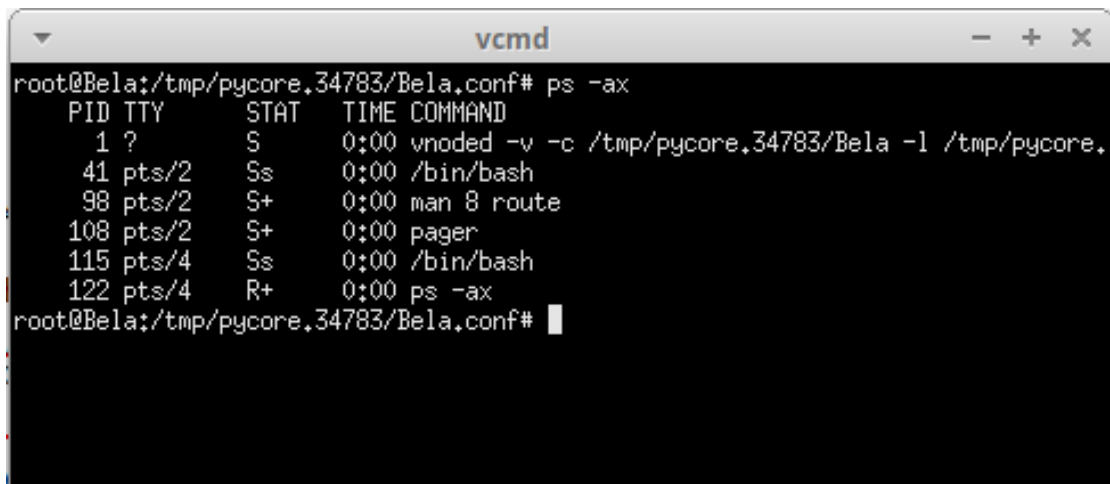
```
root@RA:/tmp/pycore.43899/RA.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt Iface
10.0.0.0         0.0.0.0          255.255.255.0   U       0 0        0 eth0
10.0.1.0         10.0.3.2         255.255.255.0   UG      0 0        0 eth1
10.0.2.0         10.0.3.2         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         10.0.3.2         255.255.255.0   UG      0 0        0 eth1
10.0.5.0         10.0.6.1         255.255.255.0   UG      0 0        0 eth2
10.0.6.0         0.0.0.0          255.255.255.0   U       0 0        0 eth2
10.0.7.0         0.0.0.0          255.255.255.0   U       0 0        0 eth3
10.0.16.0        10.0.6.1         255.255.255.0   UG      0 0        0 eth2
root@RA:/tmp/pycore.43899/RA.conf#
```

Figure 29: Tabela de encaminhamento do *router* A

Pela tabela do host Bela, podemos concluir que tem duas entradas: uma com o destino 0.0.0.0, que é o endereço *default* quando não se sabe o destino do pacote, e a outra entrada tem destination 10.0.0.0 que é usada para enviar um pacote para a própria rede. Já na tabela do *router* A, podemos verificar os destinos possíveis e o respetivo *gateway*.

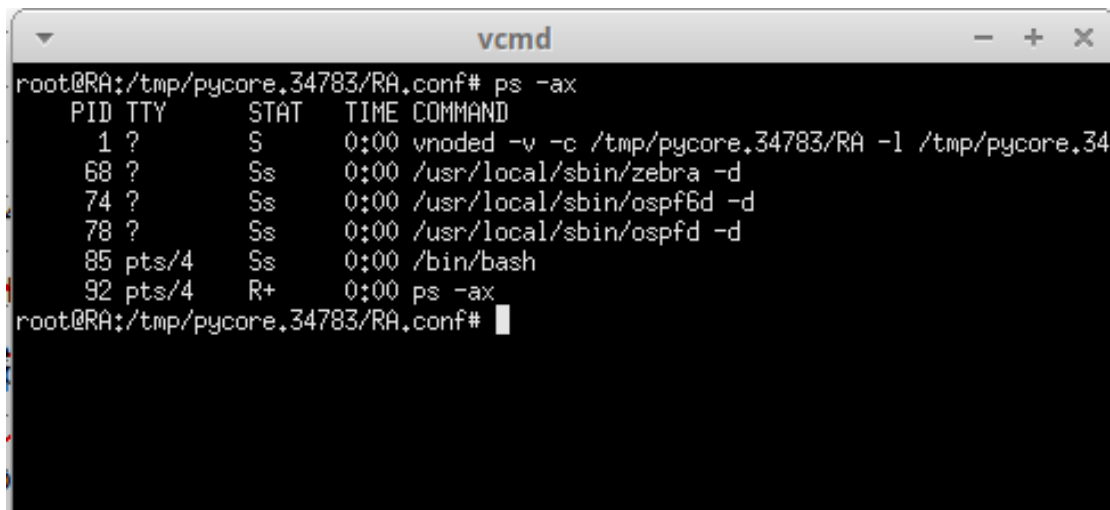
b) Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema, por exemplo, `ps -ax` ou equivalente).

Pelas imagens, podemos verificar que existem processos a correr os protocolos ZEBRA e OSPF, concluído que o router está a utilizar encaminhamento dinâmico (as rotas são atualizadas através destes protocolos).



```
vcmd
root@Bela:/tmp/pycore.34783/Bela.conf# ps -ax
  PID TTY          STAT       TIME COMMAND
    1 ?           S            0:00 vnodes -v -c /tmp/pycore.34783/Bela -l /tmp/pycore.
   41 pts/2      Ss           0:00 /bin/bash
   98 pts/2      S+           0:00 man 8 route
  108 pts/2      S+           0:00 pager
  115 pts/4      Ss           0:00 /bin/bash
  122 pts/4      R+           0:00 ps -ax
root@Bela:/tmp/pycore.34783/Bela.conf#
```

Figure 30: Output do comando *ps -aux* no Laptop Bela



```
vcmd
root@RA:/tmp/pycore.34783/RA.conf# ps -ax
  PID TTY          STAT       TIME COMMAND
    1 ?           S            0:00 vnodes -v -c /tmp/pycore.34783/RA -l /tmp/pycore.34
   68 ?           Ss           0:00 /usr/local/sbin/zebra -d
   74 ?           Ss           0:00 /usr/local/sbin/ospfd -d
   78 ?           Ss           0:00 /usr/local/sbin/ospfd -d
   85 pts/4      Ss           0:00 /bin/bash
   92 pts/4      R+           0:00 ps -ax
root@RA:/tmp/pycore.34783/RA.conf#
```

Figure 31: Output do comando *ps -aux* no Router A

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.

Eliminando a rota *default* (0.0.0.0), deixa de ser possível comunicar com o exterior da rede onde o servidor se encontra. No entanto, ainda é possível os hosts dentro da rede comunicarem com o servidor SA, dado que estão conectados por um *switch*.

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.43339/SA.conf# route add -net 10.0.1.0 netmask 255.255.255.0
root@SA:/tmp/pycore.43339/SA.conf# route add -net 10.0.2.0 netmask 255.255.255.0
root@SA:/tmp/pycore.43339/SA.conf# route add -net 10.0.7.0 netmask 255.255.255.0
root@SA:/tmp/pycore.43339/SA.conf# route add -net 10.0.16.0 netmask 255.255.255.0

```

Figure 32: Adição manual das rotas que foram eliminadas

A print não mostra a totalidade dos comandos introduzidos. Os comandos foram:

```

route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.0.1 dev eth0
route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.0.1 dev eth0
route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.0.1 dev eth0
route add -net 10.0.16.0 netmask 255.255.255.0 gw 10.0.0.1 dev eth0

```

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.43339/SA.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	10.0.0.1	255.255.255.0	UG	0	0	0	eth0
10.0.2.0	10.0.0.1	255.255.255.0	UG	0	0	0	eth0
10.0.7.0	10.0.0.1	255.255.255.0	UG	0	0	0	eth0
10.0.16.0	10.0.0.1	255.255.255.0	UG	0	0	0	eth0

Figure 33: Tabela de encaminhamento do servidor após a adição das rotas estáticas

```

root@SA:/tmp/pycore.43339/SA.conf# ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=63 time=1.27 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=63 time=0.496 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=63 time=0.589 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=63 time=0.615 ms
64 bytes from 10.0.1.1: icmp_seq=5 ttl=63 time=0.497 ms
^C
--- 10.0.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4073ms
rtt min/avg/max/mdev = 0.496/0.693/1.272/0.293 ms
root@SA:/tmp/pycore.43339/SA.conf# ping 10.0.2.1
PING 10.0.2.1 (10.0.2.1) 56(84) bytes of data.
64 bytes from 10.0.2.1: icmp_seq=1 ttl=62 time=1.33 ms
64 bytes from 10.0.2.1: icmp_seq=2 ttl=62 time=0.677 ms
64 bytes from 10.0.2.1: icmp_seq=3 ttl=62 time=0.482 ms
64 bytes from 10.0.2.1: icmp_seq=4 ttl=62 time=0.518 ms
64 bytes from 10.0.2.1: icmp_seq=5 ttl=62 time=0.682 ms
^C
--- 10.0.2.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4079ms
rtt min/avg/max/mdev = 0.482/0.738/1.333/0.308 ms
root@SA:/tmp/pycore.43339/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=1.55 ms
64 bytes from 10.0.7.1: icmp_seq=2 ttl=63 time=0.328 ms
64 bytes from 10.0.7.1: icmp_seq=3 ttl=63 time=0.469 ms
64 bytes from 10.0.7.1: icmp_seq=4 ttl=63 time=0.604 ms
64 bytes from 10.0.7.1: icmp_seq=5 ttl=63 time=0.335 ms
^C
--- 10.0.7.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4066ms
rtt min/avg/max/mdev = 0.328/0.657/1.550/0.457 ms
root@SA:/tmp/pycore.43339/SA.conf# ping 10.0.16.1
PING 10.0.16.1 (10.0.16.1) 56(84) bytes of data.
64 bytes from 10.0.16.1: icmp_seq=1 ttl=63 time=1.28 ms
64 bytes from 10.0.16.1: icmp_seq=2 ttl=63 time=0.329 ms
64 bytes from 10.0.16.1: icmp_seq=3 ttl=63 time=0.505 ms
64 bytes from 10.0.16.1: icmp_seq=4 ttl=63 time=0.392 ms
64 bytes from 10.0.16.1: icmp_seq=5 ttl=63 time=1.11 ms
^C
--- 10.0.16.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4074ms
rtt min/avg/max/mdev = 0.329/0.723/1.281/0.393 ms

```

Figure 34: pings para os departamentos exteriores garantindo a acessibilidade do servidor

---

### 2.3 Exercício 3

Considere a topologia definida anteriormente. Assuma que o endereçamento entre os routers (rede de backbone) se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.

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 (PLXXX). 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 planejamento.

Atendendo a que o número do nosso grupo é 40, o nosso endereço de IP é 192.168.040.128/25, correspondente ao binário 11000000.10101000.00101000.10000000.

Como temos que ter pelo menos 4 sub-redes, uma para cada departamento e que existirão mais num futuro próximo, optamos por reservar 3 *bits* para a identificação da sub-rede.

Como 3 *bits* estão reservados para a identificação da sub-rede de cada departamento, sobram 4 *bits* para os *hosts* internos ao departamento.

Na tabela seguinte, estão apresentados as sub-redes que associamos a cada departamento.

000	Livre	Departamento A
001	Livre	Departamento B
010	Livre	Departamento C
011	Livre	Departamento D
100	Livre	
101	Livre	
110	Livre	
111	Livre	

Table 2: Sub-rede ID associado a cada departamento

Departamento	Rede Gerada	Intervalos de Interface Possíveis
Departamento A	192.168.040.128	192.168.040.129 a 192.040.142
Departamento B	192.168.040.144	192.168.040.145 a 192.040.158
Departamento C	192.168.040.160	192.168.040.161 a 192.040.174
Departamento D	192.168.040.176	192.168.040.177 a 192.040.190

Table 3: Endereços de cada departamento e *hosts*

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.

Como são necessários, pelo menos 4 departamentos, e atendendo a que o número irá



aumenta em breve, precisamos de reservar 3 *bits* para a identificação de cada um deles, fazendo com que tenhamos 8 sub-redes, uma para cada departamento. Posto isto, a nossa máscara de rede deixa de ser /25 e passa para /28 (**255.255.255.240**).

Como nos sobram 4 *bits* para *hosts*, podemos interligar  $2^4 - 2 = 14$  *hosts* dentro de cada departamento.

Ficam 4 prefixos de sub-rede disponíveis para uso futuro, uma vez que só fizemos *subnetting* dos 4 departamentos que existem atualmente.

**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.**

Primeiramente fizemos a alteração dos IPs dos computadores, servidores e *routers* dos departamentos para os novos endereços de *subnetting*. De seguida, para testar a conectividade interna na rede recorremos ao comando *ping* a partir de um computador do departamento A para um computador de cada um dos outros departamentos.

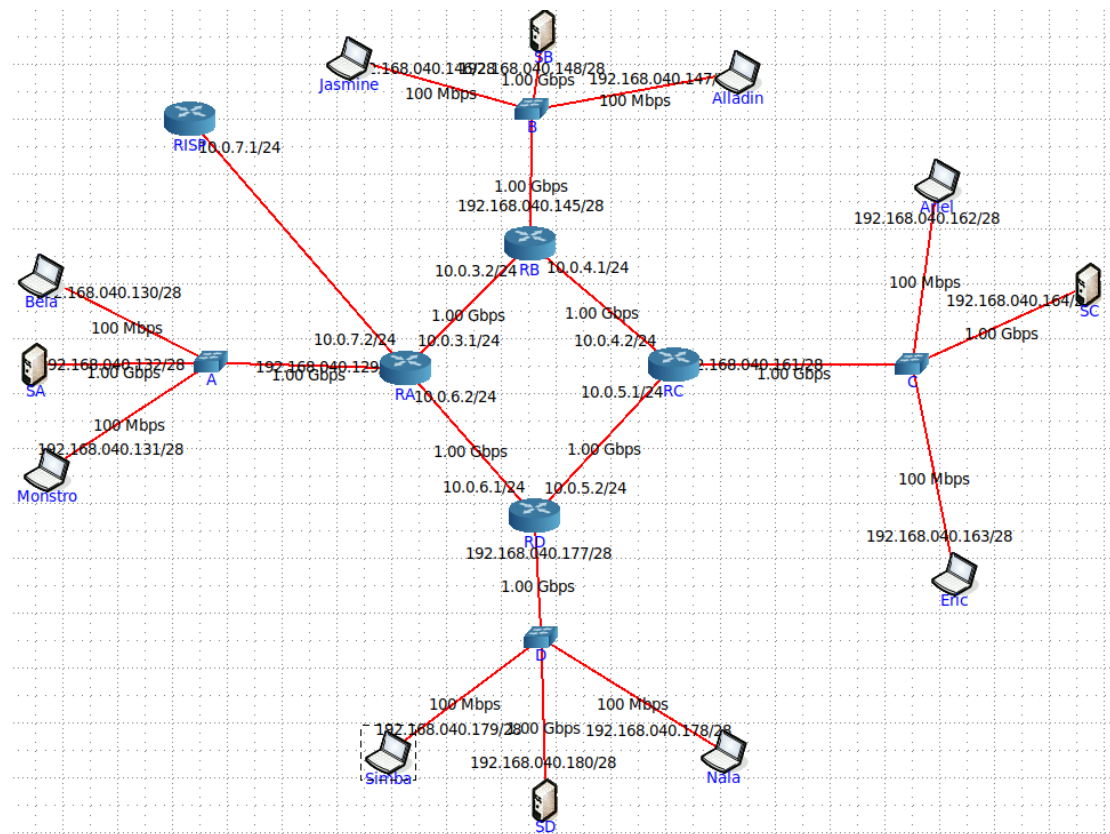


Figure 35: Topologia após alteração dos IPs

```

root@Bela:/tmp/pycore.43339/Bela.conf# ping 192.168.040.146
PING 192.168.040.146 (192.168.32.146) 56(84) bytes of data.
64 bytes from 192.168.32.146: icmp_seq=1 ttl=62 time=0.866 ms
64 bytes from 192.168.32.146: icmp_seq=2 ttl=62 time=0.536 ms
64 bytes from 192.168.32.146: icmp_seq=3 ttl=62 time=0.540 ms
64 bytes from 192.168.32.146: icmp_seq=4 ttl=62 time=0.829 ms
64 bytes from 192.168.32.146: icmp_seq=5 ttl=62 time=0.821 ms
^C
--- 192.168.040.146 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4102ms
rtt min/avg/max/mdev = 0.536/0.718/0.866/0.148 ms
root@Bela:/tmp/pycore.43339/Bela.conf# ping 192.168.040.162
PING 192.168.040.162 (192.168.32.162) 56(84) bytes of data.
64 bytes from 192.168.32.162: icmp_seq=1 ttl=61 time=1.03 ms
64 bytes from 192.168.32.162: icmp_seq=2 ttl=61 time=0.655 ms
64 bytes from 192.168.32.162: icmp_seq=3 ttl=61 time=1.02 ms
64 bytes from 192.168.32.162: icmp_seq=4 ttl=61 time=1.22 ms
64 bytes from 192.168.32.162: icmp_seq=5 ttl=61 time=0.467 ms
^C
--- 192.168.040.162 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4014ms
rtt min/avg/max/mdev = 0.467/0.878/1.217/0.274 ms
root@Bela:/tmp/pycore.43339/Bela.conf# ping 192.168.040.179
PING 192.168.040.179 (192.168.32.179) 56(84) bytes of data.
64 bytes from 192.168.32.179: icmp_seq=1 ttl=62 time=0.936 ms
64 bytes from 192.168.32.179: icmp_seq=2 ttl=62 time=1.02 ms
64 bytes from 192.168.32.179: icmp_seq=3 ttl=62 time=1.04 ms
64 bytes from 192.168.32.179: icmp_seq=4 ttl=62 time=0.646 ms
64 bytes from 192.168.32.179: icmp_seq=5 ttl=62 time=0.472 ms
^C
--- 192.168.040.179 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4034ms
rtt min/avg/max/mdev = 0.472/0.823/1.042/0.225 ms
root@Bela:/tmp/pycore.43339/Bela.conf# █

```

Figure 36: *Ping* do portátil Bela para os portáteis Jasmine, Ariel e Simba, respetivamente

---

### 3 Conclusão

Na primeira fase deste trabalho abordamos a transmissão de dados referente a máquinas dentro da mesma rede e a necessidade, ou não, de fragmentação no envio de pacotes de dados.

Na segunda fase abordamos o funcionamento do encaminhamento e endereçamento entre vários departamentos distintos, cada um com a sua sub-rede, e a diferença entre os dois tipos de encaminhamento e formas de endereçar redes.

Em suma, este trabalho foi bastante importante para consolidar os diversos conhecimentos adquiridos ao longo das aulas teóricas.