

# Universidade do Minho

Licenciatura em Engenharia Informática

## Redes de Computadores

Trabalho Prático 2

### Grupo 101

Ana Murta (A93284)  
Ana Henriques (A93268)  
Leonardo Freitas (A93281)

abril, 2022

# Conteúdo

<b>1 Parte I - Datagramas IP e Fragmentação</b>	<b>5</b>
1.1 Exercício 1 . . . . .	5
1.1.1 Questão a . . . . .	6
1.1.2 Questão b . . . . .	6
1.1.3 Questão c . . . . .	7
1.1.4 Questão d . . . . .	7
1.1.5 Questão e . . . . .	7
1.2 Exercício 2 . . . . .	8
1.2.1 Questão a . . . . .	8
1.2.2 Questão b . . . . .	8
1.2.3 Questão c . . . . .	8
1.2.4 Questão d . . . . .	9
1.2.5 Questão e . . . . .	9
1.2.6 Questão f . . . . .	10
1.2.7 Questão g . . . . .	11
1.3 Exercício 3 . . . . .	12
1.3.1 Questão a . . . . .	12
1.3.2 Questão b . . . . .	13
1.3.3 Questão c . . . . .	13
1.3.4 Questão d . . . . .	14
1.3.5 Questão e . . . . .	14
1.3.6 Questão f . . . . .	14
1.3.7 Questão g . . . . .	15
<b>2 Parte II - Endereçamento e Encaminhamento IP</b>	<b>16</b>
2.1 Exercício 1 . . . . .	16
2.1.1 Questão a . . . . .	16
2.1.2 Questão b . . . . .	17
2.1.3 Questão c . . . . .	17
2.1.4 Questão d . . . . .	17
2.1.5 Questão e . . . . .	18
2.1.6 Questão f . . . . .	19
2.2 Exercício 2 . . . . .	20
2.2.1 Questão a . . . . .	20
2.2.2 Questão b . . . . .	21
2.2.3 Questão c . . . . .	21
2.2.4 Questão d . . . . .	22
2.2.5 Questão e . . . . .	22
2.3 Definição de Sub-redes . . . . .	24
2.3.1 Questão 1 . . . . .	24
2.3.2 Questão 2 . . . . .	25
2.3.3 Questão 3 . . . . .	25



# Listas de Figuras

1.1	Topologia CORE . . . . .	5
1.2	Shell de Bela . . . . .	6
1.3	Tráfego enviado e recebido por Bela . . . . .	6
1.4	Valor mínimo de TTL para alcançar Monstro . . . . .	7
1.5	Tempo de ida e volta . . . . .	7
1.6	Wireshark <i>tab</i> da primeira mensagem ICMP capturada . . . . .	8
1.7	Tamanho do campo de dados do datagrama . . . . .	9
1.8	Campo <i>flags</i> do datagrama IP capturado . . . . .	9
1.9	Tráfego ICMP gerado a partir do IP atribuído à interface do nosso computador . . . . .	10
1.10	Comparação dos campos de cabeçalho dos pacotes 57 e 60 . . . . .	10
1.11	Tráfego ICMP do datagrama IP capturado . . . . .	10
1.12	Mensagens <i>Time to live exceeded in transit</i> . . . . .	11
1.13	Tráfego ICMP recebido pela nossa máquina . . . . .	11
1.14	Fragmentação do pacote inicial . . . . .	12
1.15	Primeiro fragmento do datagrama IP segmentado . . . . .	13
1.16	Segundo fragmento do datagrama IP segmentado . . . . .	13
1.17	Equipamentos existentes . . . . .	14
2.1	Topologia LEI-RC . . . . .	17
2.2	Teste à conectividade IP interna de cada departamento. . . . .	18
2.3	Teste IP entre os varios Departamentos. . . . .	18
2.4	Teste IP entre a Bela e o $R_{ISP}$ . . . . .	19
2.5	Tabela de encaminhamento do router RA . . . . .	20
2.6	Tabela de encaminhamento do portátil Bela . . . . .	20
2.7	Tabela de encaminhamento do portátil Bela da rede A . . . . .	21
2.8	Execução do comando <code>route delete default</code> para o servidor SA . . . . .	21
2.9	Adição das rotas estáticas ao servidor SA . . . . .	22
2.10	<code>ping 10.0.8.10</code> . . . . .	22
2.11	<code>ping 10.0.3.10</code> . . . . .	22
2.12	<code>ping 10.0.0.10</code> . . . . .	22
2.13	<code>ping 10.0.6.10</code> . . . . .	22
2.14	Tabela de encaminhamento do servidor SA . . . . .	23
2.15	Esquema de endereçamento . . . . .	24
2.16	Teste à conetividade IP interna na rede local LEI-RC . . . . .	25

# Capítulo 1

## Parte I - Datagramas IP e Fragmentação

### 1.1 Exercício 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 estabilize.

Na figura 2.1, está, então, ilustrada a topologia CORE desenvolvida para verificar o comportamento do *traceroute*, respeitando todos os requisitos estipulados.

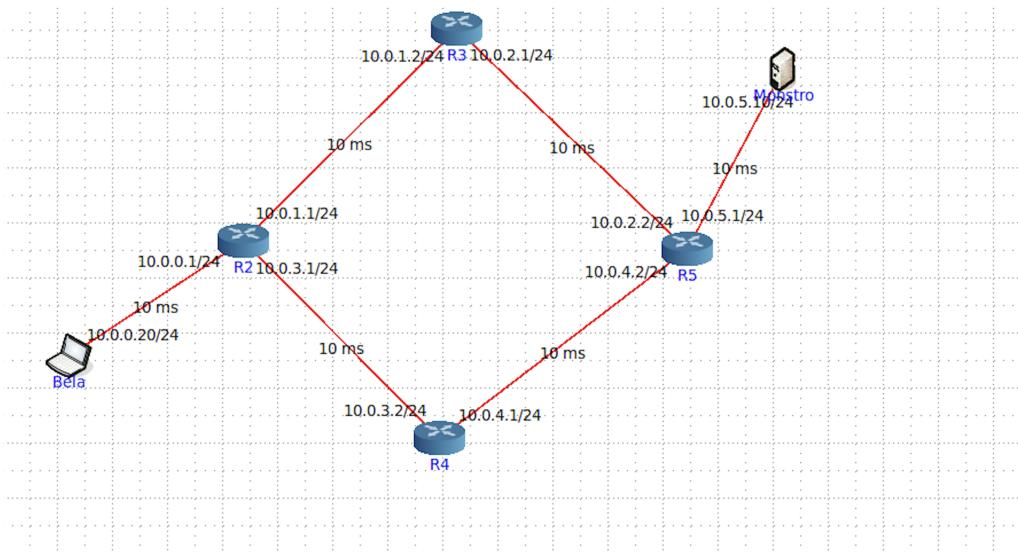


Figura 1.1: Topologia CORE

### 1.1.1 Questão 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.

O endereço IP do Monstro é 10.0.5.10. Na Figura 1.2, temos o resultado da execução do comando `traceroute -I` 10.0.5.10.

```

root@Bela:/tmp/pycore.36741/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)  40.940 ms  40.898 ms  40.896 ms
 2  10.0.1.2 (10.0.1.2)  61.769 ms  61.770 ms  61.770 ms
 3  10.0.2.2 (10.0.2.2)  103.735 ms  103.736 ms  103.737 ms
 4  10.0.5.10 (10.0.5.10)  145.362 ms  145.363 ms  145.362 ms
root@Bela:/tmp/pycore.36741/Bela.conf#

```

Figura 1.2: Shell de Bela

### 1.1.2 Questão 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.

Observamos pela Figura 1.3 que, do cliente Bela, foram enviados vários pacotes. Os primeiros pacotes tinham TTL = 1 e foram descartados por R2. De seguida, foram enviados pacotes com TTL = 2 também descartados, desta vez por R3. Posteriormente, foram enviados ainda mais pacotes com TTL = 3 também descartados, desta vez por R5. Por fim, foram enviados pacotes com TTL = 4 que finalmente chegaram ao destino. Por cada pacote descartado, foi recebida uma resposta proveniente do router que o descartou “*Time to live exceeded*”.

39 051513253 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=1/765, ttl=1 (no response..)
39 0515132615 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=2/712, ttl=1 (no response..)
32 39 0515145691 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=3/768, ttl=1 (no response..)
33 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=4/1024, ttl=2 (no response..)
34 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=5/1280, ttl=2 (no response..)
35 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=6/1536, ttl=2 (no response..)
36 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=7/1792, ttl=2 (no response..)
37 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=8/2048, ttl=3 (no response..)
38 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=9/2304, ttl=3 (no response..)
39 0515162873 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=10/2560, ttl=4 (reply in ..)
40 39 0515162873 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) reply id=0x0018 seq=11/2816, ttl=4 (request ..)
41 39 0515169527 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=12/3072, ttl=4 (reply in ..)
42 39 0515169978 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=13/3328, ttl=5 (reply in ..)
43 39 0515118480 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=14/3584, ttl=5 (reply in ..)
44 39 0515118480 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=15/3840, ttl=5 (reply in ..)
45 39 0515118480 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=16/4096, ttl=6 (reply in ..)
46 39 0515144867 10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
47 39 0515144867 10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
48 39 0515144867 10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
49 39 0379744762 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=17/4352, ttl=6 (reply in ..)
50 39 0379798884 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=18/4608, ttl=6 (reply in ..)
51 39 0379798884 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=19/4864, ttl=6 (reply in ..)
52 39 0515159561 10.0.0.20	10.0.5.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
53 39 0515159561 10.0.0.20	10.0.5.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
54 39 0515159561 10.0.0.20	10.0.5.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
55 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=20/5120, ttl=7 (reply in ..)
56 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=21/5376, ttl=7 (reply in ..)
57 39 0515159561 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=22/5632, ttl=8 (reply in ..)
58 39 0515159561 10.0.0.20	10.0.5.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
59 39 0515159561 10.0.0.20	10.0.5.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
60 39 0515159561 10.0.0.20	10.0.5.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
61 39 101377609 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=23/5888, ttl=8 (reply in ..)
62 39 101397851 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=24/6144, ttl=8 (reply in ..)
63 39 101409277 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x0018 seq=25/6400, ttl=8 (reply in ..)
64 39 101409277 10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) reply id=0x0018 seq=26/6656, ttl=8 (request ..)
65 39 149238838 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=11/2816, ttl=6 (request ..)
66 39 149238838 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=12/3072, ttl=6 (request ..)
67 39 149241144 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=13/3328, ttl=6 (request ..)
68 39 149241144 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=14/3584, ttl=6 (request ..)
69 39 149242921 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=15/3840, ttl=6 (request ..)
70 39 149243987 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=16/4096, ttl=6 (request ..)
71 39 149245119 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=17/4352, ttl=6 (request ..)
72 39 149245119 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=18/4608, ttl=6 (request ..)
73 39 149247301 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=19/4864, ttl=6 (request ..)
74 39 149248196 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=20/5120, ttl=6 (request ..)
75 39 149249225 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=21/5376, ttl=6 (request ..)
76 39 149249225 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=22/5632, ttl=6 (request ..)
77 39 184186943 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=23/5888, ttl=6 (request ..)
78 39 184282698 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=24/6144, ttl=6 (request ..)
79 39 184285081 10.0.5.19	10.0.0.20	ICMP	74 Echo (ping) reply id=0x0018 seq=25/6400, ttl=6 (request ..)

Figura 1.3: Tráfego enviado e recebido por Bela

### 1.1.3 Questão 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.

Através da Figura 1.4, é possível concluir que o valor mínimo de TTL para alcançar o servidor Monstro é TTL = 61. Isto também pode ser comprovado pela Figura 1.3, em que Bela só recebe uma resposta de Monstro quando TTL = 61.

Time (ms)	Source IP	Destination IP	TTL	Type	Code	Info
01 39 101377899	10.0.0.29	10.0.5.10	ICMP	74 Echo (ping) request	id=8x018, seq=23/5888, ttl=1 (request in 77)	
02 39 101397859	10.0.0.29	10.0.5.10	ICMP	74 Echo (ping) request	id=8x018, seq=24/6144, ttl=9 (request in 78)	
03 39 101409277	10.0.0.29	10.0.5.10	ICMP	74 Echo (ping) request	id=8x018, seq=25/6400, ttl=9 (request in 79)	
04 39 101410274	10.0.0.29	10.0.5.10	ICMP	74 Echo (ping) request	id=8x018, seq=26/6656, ttl=9 (request in 80)	
05 39 140238838	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=11/2816, ttl=61 (request in 48)	
06 39 140239958	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=12/3972, ttl=61 (request in 41)	
07 39 140240068	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=13/5128, ttl=61 (request in 45)	
08 39 140242943	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=14/3584, ttl=61 (request in 49)	
09 39 140242921	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=15/3840, ttl=61 (request in 44)	
10 39 140243987	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=16/4996, ttl=61 (request in 45)	
11 39 140244000	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=17/6152, ttl=61 (request in 49)	
12 39 140246021	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=18/4688, ttl=61 (request in 50)	
13 39 140247381	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=19/4864, ttl=61 (request in 51)	
14 39 140248320	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=20/5020, ttl=61 (request in 55)	
15 39 140253225	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=21/5376, ttl=61 (request in 56)	
16 39 140258122	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=22/5632, ttl=61 (request in 57)	
17 39 184186948	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=23/5888, ttl=61 (request in 61)	
18 39 184262093	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=24/6144, ttl=61 (request in 62)	
19 39 184269013	10.0.5.10	10.0.0.29	ICMP	74 Echo (ping) reply	id=8x018, seq=25/6400, ttl=61 (request in 63)	

Figura 1.4: Valor mínimo de TTL para alcançar Monstro

### 1.1.4 Questão 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 tempo de ida-e-volta obtido no acesso ao servidor é  $(145.362 + 145.363 + 145.362)/3 = 145.362$  – Figura 1.5.

```
vcmd
root@Bela:/tmp/pycore.36741/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)  40.940 ms  40.898 ms  40.896 ms
 2  10.0.1.2 (10.0.1.2)  61.769 ms  61.770 ms  61.770 ms
 3  10.0.2.2 (10.0.2.2)  103.735 ms  103.736 ms  103.737 ms
 4  10.0.5.10 (10.0.5.10)  145.362 ms  145.363 ms  145.362 ms
root@Bela:/tmp/pycore.36741/Bela.conf#
```

Figura 1.5: Tempo de ida e volta

### 1.1.5 Questão 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 número de nodos é a razão pela qual não podemos calcular o valor médio do atraso num sentido desta forma, dada a possibilidade de haver vários percursos. Outro impedimento também pode ser o congestionamento na rede.

## 1.2 Exercício 2

**Exercício 2.** Pretende-se agora usar o traceroute na sua máquina nativa e gerar datagramas IP de diferentes tamanhos. Usando o wireshark capture o tráfego gerado pelo traceroute para os seguintes tamanhos de pacote: situação (i) sem especificar, i.e., usando o tamanho do pacote de prova por defeito; e situação (ii) ( $4000 + X$ ) bytes, em que  $X$  é o número do grupo de trabalho. Utilize como máquina destino o host marco.uminho.pt. Pare a captura. Com base no tráfego capturado, identifique os pedidos ICMP Echo Request e o conjunto de mensagens devolvidas como resposta. Selecione a primeira mensagem ICMP capturada (referente à situação (i) tamanho por defeito) e centre a análise no nível protocolar IP (expanda o tab correspondente na janela de detalhe do wireshark).

### 1.2.1 Questão a

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

Através da 1.6, inferimos que o endereço IP da interface ativa é 172.26.5.148

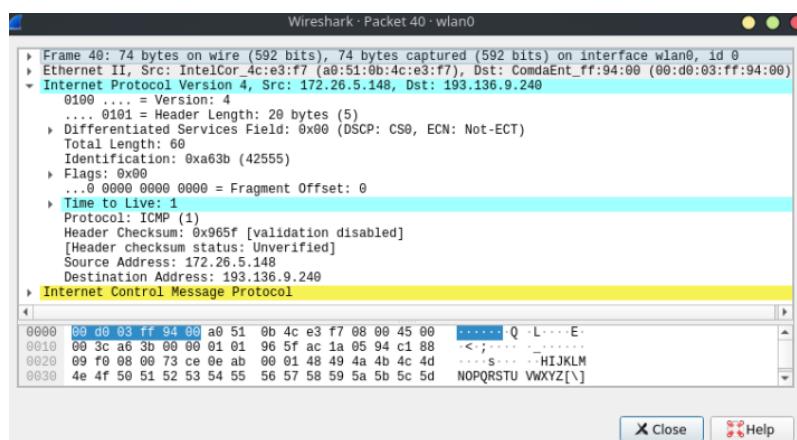


Figura 1.6: Wireshark tab da primeira mensagem ICMP capturada

### 1.2.2 Questão b

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

Como podemos ver pela Figura 1.6, o campo protocolo tem como valor 01, identificando, assim, o ICMP – “IP Protocol: ICMP(1)”.

### 1.2.3 Questão c

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

O campo de dados do datagrama tem 40 bytes (Figura 1.7); isto porque:

```
Header Length: 20 bytes
Total Length: 60 bytes
Payload Length = Total Length - Header Length = 60 - 20 = 40 bytes
```

```

> Frame 41: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlan0, id 0
> Ethernet II, Src: IntelCor_4c:e3:f7 (a0:51:0b:4c:e3:f7), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.5.148, Dst: 193.136.9.240
    0100 . . . . . Version: 4
    0101 . . . . . Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSSCP: CS0, ECN: Not-ECT)
        Total Length: 60
        Identification: 0xa63c (42556)
        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: 0x065e [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.5.148
    Destination Address: 193.136.9.240
    > Internet Control Message Protocol

```

Figura 1.7: Tamanho do campo de dados do datagrama

#### 1.2.4 Questão d

**O datagrama IP foi fragmentado? Justifique.**

Na Figura 1.8, temos “*Fragment Offset: 0*”, o que indica que estamos no início do ficheiro e temos, ainda, “*More fragments: Not set*”, o que prova que é também a última parte do pacote. Como tal, o datagrama IP não está fragmentado.

```

> Frame 60: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlan0, id 0
> Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: IntelCor_4c:e3:f7 (a0:51:0b:4c:e3:f7)
> Internet Protocol Version 4, Src: 193.136.9.240, Dst: 172.26.5.148
    0100 . . . . . Version: 4
    0101 . . . . . Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSSCP: CS0, ECN: Not-ECT)
        Total Length: 60
        Identification: 0x0fd1 (4049)
        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: 61
    Protocol: ICMP (1)
    Header Checksum: 0xf0c9 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 193.136.9.240
    Destination Address: 172.26.5.148
    > Internet Control Message Protocol
        Type: 0 (Echo (ping) request)
        Code: 0
        Checksum: 0xb7c5 [correct]
        [Checksum Status: Good]
        Identifier (LE): 43790 (0xab0b)
        Identifier (LE): 43790 (0xab0b)
        Sequence Number (BE): 10 (0x000a)
        Sequence Number (BE): 2560 (0x8a00)
        [Request frame: 49]
        [Response time: 2.048 ms]
        Data (32 bytes)
            Data: 4849444b4c4d4e4f505152535455565758595a5b5c5d5e5f6001626364656667
            [Length: 32]

```

Figura 1.8: Campo *flags* do datagrama IP capturado

#### 1.2.5 Questão 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.

Na Figura 1.9, podemos analisar a sequência de tráfego ICMP gerado a partir do endereço IP atribuído à interface da nossa máquina.

No.	Time	Source	Destination	Protocol	Length	Info
70	6.578419977	172.16.115.252	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
71	6.578420000	172.16.115.252	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
72	6.579226791	172.16.115.252	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
73	6.579226814	172.16.115.252	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
56	6.577762149	172.16.2.1	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
59	6.578418688	172.16.2.1	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
60	6.578418701	172.16.2.1	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
59	6.578418688	172.26.254.254	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
63	6.578418166	172.26.254.254	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
64	6.578418169	172.26.254.254	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
49	6.576328649	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=1/256, ttl=1 (no response found!)
41	6.576343514	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=2/512, ttl=1 (no response found!)
42	6.576347330	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=3/768, ttl=1 (no response found!)
43	6.576347333	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=4/1024, ttl=1 (no response found!)
44	6.576347336	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=5/1289, ttl=2 (no response found!)
45	6.576357731	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=6/1536, ttl=2 (no response found!)
47	6.576357734	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=7/2048, ttl=3 (no response found!)
48	6.576357744	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=8/2496, ttl=3 (no response found!)
49	6.576357745	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=9/2384, ttl=3 (no response found!)
50	6.576357746	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=10/2560, ttl=4 (reply in 60)
51	6.576357747	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=11/2872, ttl=4 (reply in 61)
52	6.576357748	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=12/3072, ttl=4 (reply in 64)
53	6.576357749	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=13/3228, ttl=5 (reply in 65)
54	6.576357750	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=14/3584, ttl=5 (reply in 67)
55	6.576357751	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=15/3840, ttl=5 (reply in 68)
56	6.576357752	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=16/4096, ttl=5 (reply in 69)
57	6.577836461	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=17/4352, ttl=6 (reply in 71)
58	6.577836462	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=18/4608, ttl=6 (reply in 74)
60	6.578419827	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=11/2816, ttl=61 (request in 49)
61	6.578419852	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=12/3072, ttl=61 (request in 51)
65	6.578419878	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=13/3228, ttl=61 (request in 52)
67	6.578419901	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=14/3584, ttl=61 (request in 53)
68	6.578419927	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=15/3840, ttl=61 (request in 54)
69	6.578419952	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=16/4096, ttl=61 (request in 55)
74	6.579538423	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=17/4352, ttl=61 (request in 57)

Figura 1.9: Tráfego ICMP gerado a partir do IP atribuído à interface do nosso computador

A partir da Figura 1.10, concluímos que os campos do cabeçalho IP que variam de pacote para pacote são: *Identification*, *Time to Live* e *Header checksum*.

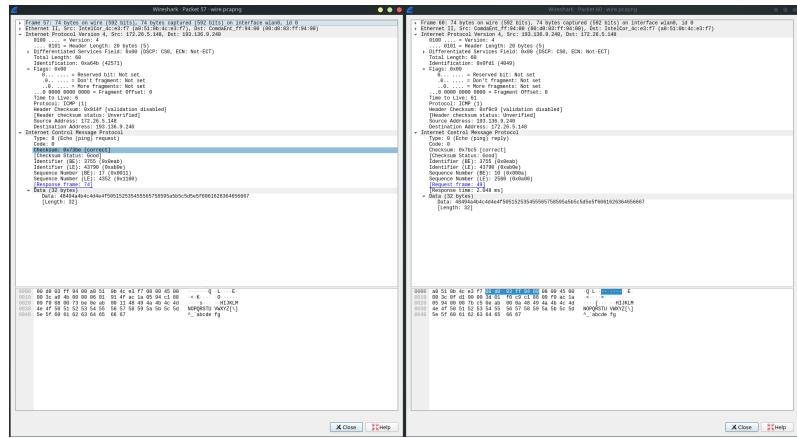


Figura 1.10: Comparação dos campos de cabeçalho dos pacotes 57 e 60

## 1.2.6 Questão f

Observa um padrão nos valores do campo de identificação do datagrama IP e TTL?

Relativamente ao campo de identificação do datagrama IP, este tem os 8 bits iguais. Relativamente ao TTL, são lançados pacotes com valores de TTL cada vez maiores até ao momento em que um deles chega ao destino. Isto pode ser confirmado pela Figura 1.11.

No.	Time	Source	Destination	Protocol	Length	Info
49	6.576328649	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=1/256, ttl=1 (no response found!)
42	6.576347330	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=2/512, ttl=1 (no response found!)
43	6.576351507	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=4/1024, ttl=2 (no response found!)
44	6.576354788	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=5/1289, ttl=2 (no response found!)
45	6.576361369	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=6/1536, ttl=3 (no response found!)
46	6.576361370	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=7/2048, ttl=3 (no response found!)
47	6.576361371	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=8/2496, ttl=3 (no response found!)
48	6.576361372	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=9/2384, ttl=3 (no response found!)
49	6.576374385	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=10/2560, ttl=4 (reply in 60)
50	6.576374386	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=11/2816, ttl=4 (reply in 61)
51	6.576377463	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=12/3072, ttl=4 (reply in 64)
52	6.576384242	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=13/3228, ttl=5 (reply in 65)
53	6.576384243	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=14/3584, ttl=5 (reply in 67)
54	6.576387418	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=15/3840, ttl=5 (reply in 68)
55	6.576387419	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) request id=0xebab, seq=16/4096, ttl=5 (reply in 69)
57	6.577836461	172.26.5.148	193.136.9.248	ICMP	74	Echo (ping) reply id=0xebab, seq=10/2560, ttl=61 (request in 49)
60	6.578419749	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=10/2560, ttl=61 (request in 50)
61	6.578419827	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=11/2816, ttl=61 (request in 51)
65	6.578419878	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=12/3072, ttl=61 (request in 52)
67	6.578419901	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=13/3228, ttl=61 (request in 53)
69	6.578419952	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=14/3584, ttl=61 (request in 54)
74	6.579538423	193.136.9.248	172.26.5.148	ICMP	74	Echo (ping) reply id=0xebab, seq=15/3840, ttl=61 (request in 55)

Figura 1.11: Tráfego ICMP do datagrama IP capturado

### 1.2.7 Questão 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?

Pela Figura 1.13, no Wireshark *tab* do pacote 66, temos TTL = 255. No entanto, este valor vai decrementando 1 unidade à medida que faz saltos de router para router. Assim sendo, o valor de TTL varia entre 253 e 255 visto que, como passa por 3 routers intermédios, este decremente 3 vezes. É importante denotar, porém, que isto também depende da marca do fabricante.

No.	Time	Source	Destination	Protocol	Length	Info
71	6.5/8419977	172.16.115.252	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
72	6.5/8419978	172.16.115.252	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
50	6.5/77762140	172.16.115.252	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
58	6.5/78416868	172.16.2.1	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
62	6.5/78417044	172.16.2.1	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
59	6.5/78418088	172.26.254.254	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
63	6.5/78418186	172.26.254.254	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
66	6.5/78418191	172.26.254.254	172.26.5.148	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)

Figura 1.12: Mensagens *Time to live exceeded in transit*

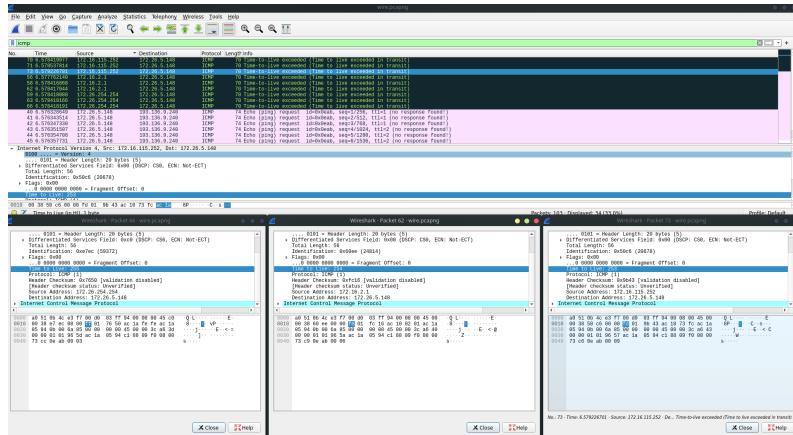


Figura 1.13: Tráfego ICMP recebido pela nossa máquina

## 1.3 Exercício 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.

### 1.3.1 Questão a

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

O pacote inicial enviado tem 4101 bytes. No entanto, a mensagem teve de ser fragmentada porque o seu tamanho era demasiado grande para poder ser transferida de uma vez. Tal como podemos ver pela Figura 1.14, o *Max Transfer Unit* (MTU) é 1500 bytes: 20 para o cabeçalho e 1480 para os dados. A sua fragmentação em pacotes mais pequenos possibilitou o seu transporte.

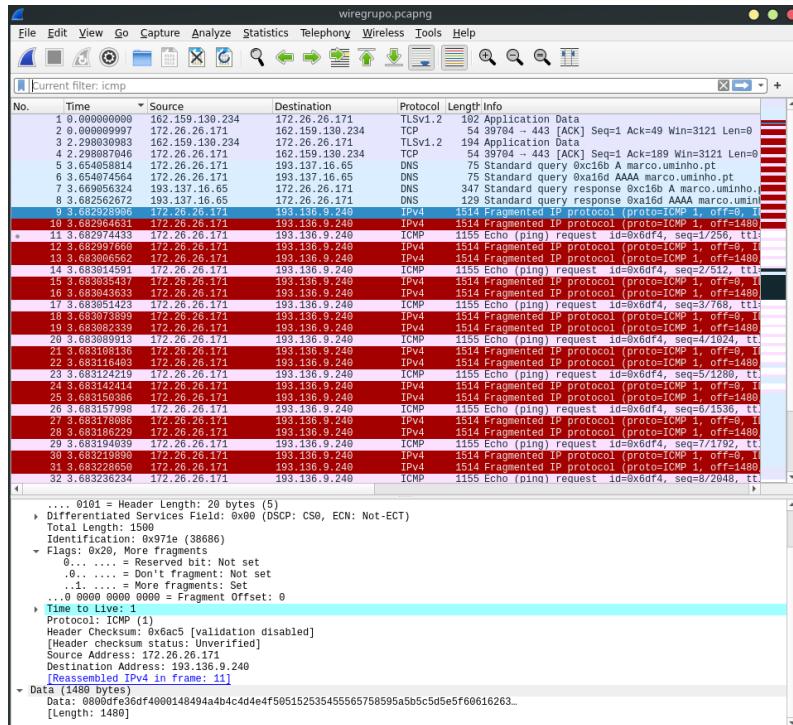


Figura 1.14: Fragmentação do pacote inicial

### 1.3.2 Questão 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?

Na figura 1.15, temos o *offset* a 0 no cabeçalho, o que indica que se trata do primeiro fragmento do pacote. Já a *flag More fragments* toma o valor 1, o que indica que este fragmento é seguido de mais fragmentos do pacote que lhe deu origem. Como o cabeçalho tem 20 bytes e a parte dos dados tem 1480 bytes, o datagrama IP tem 1500 bytes.

```
Frame 9: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlan0, id 0
  Ethernet II, Src: IntelCor_4c:83:f7 (ab:51:0b:4c:83:f7), Dst: ComdaEnt_ff:94:00 (00:00:03:ff:94:00)
  Internet Protocol Version 4, Src: 172.26.26.171, Dst: 193.136.9.249
    IP Header:
      Version = 4
      ...0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x90 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 000000000748 (38686)
    Identification: 0000000000000000 = Fragment Offset: 0
    Flags: More Fragments
      .0..... = Reserved bit: Not set
      .0..... = Don't fragment: Not set
      .1..... = More fragments: Set
      ...0000 0000 0000 = Fragment Offset: 0
  Time to Live: 128
  Protocol: ICMP (1)
  Header Checksum: 0x6ac5 [validation disabled]
  Header Checksum Status: Unverified
  Source Address: 172.26.26.171
  Destination Address: 193.136.9.249
  [Reassembled IPv4 in frame: 11]
  Data (1480 bytes)
  Data: 8980dfe83d0f4000148494a4b4cd4e4f505152535455565758595a5b5c5d5e5f60616263...
  [Length: 1480]
```

Figura 1.15: Primeiro fragmento do datagrama IP segmentado

### 1.3.3 Questão 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?

Na figura 1.16, temos que o *offset* é 1480, ou seja, é diferente de 0, podendo, assim, concluir que este não é o primeiro fragmento do datagrama IP original. De facto, isto significa que já se transferiu cerca de 1480 bytes do pacote inicial. Para além disto, como a *flag More fragments* toma o valor de 1, averiguamos que ainda há mais fragmentos.

```
> Frame 10: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlan0, id 0
> Ethernet II, Src: IntelCor_4c:e3:f7 (a0:51:00:4c:e3:f7), Dst: ComdaEnt_ff:94:00 (00:d0:00:ff:94:00)
-> Internet Protocol Version 4, Src: 172.26.26.171, Dst: 193.136.9.240
  0100 ..0.. Version: 4
    .0111 ... Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x08 (DSCP: CS8, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0x971e (38686)
    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: 128
  Protocol: ICMP (1)
  Header Checksum: 0x6a0c [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.26.171
  Destination Address: 193.136.9.240
  [Reassembled IPv4 in Frame: 11]
-> Data (1480 bytes)
  Data: 484944ab4c4d4e4f 0f0515253545556758595a5b5c5d5e5f606162636465666768696a6b...
  [Length: 1480]
```

Figura 1.16: Segundo fragmento do datagrama IP segmentado

### 1.3.4 Questão d

Quantos fragmentos foram criados a partir do datagrama original?

No final da Figura 1.17, temos “3 IPv4 Fragments (4081 bytes): #94(1480), #95(1480), #95(1480), #96(1121)”. Isto significa que, a partir do datagrama original, foram criados 3 fragmentos.

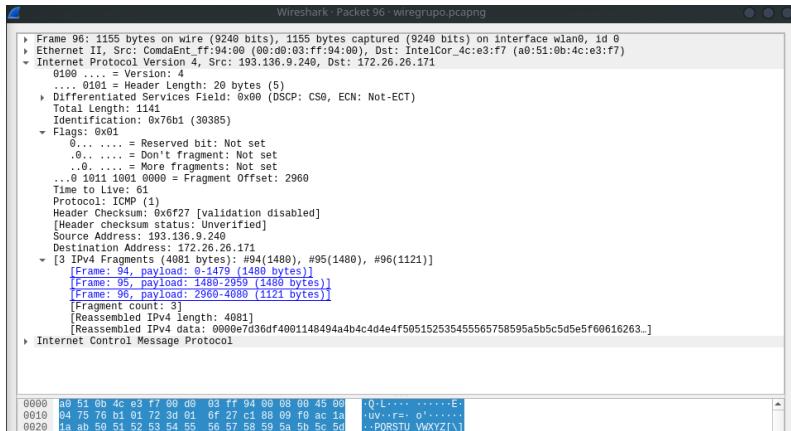


Figura 1.17: Equipamentos existentes

### 1.3.5 Questão 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 campos que mudam no cabeçalho IP, e, consequentemente, permitem a identificação do último fragmento, são:

- **length:** enquanto que este é máximo nos outros fragmentos, o último fragmento pode não atingir esse valor máximo;
- **flag More fragments:** este toma o valor 1 em todos os fragmentos menos no último;
- **offset:** este corresponde ao número de bytes que já foram enviados do datagrama inicial. Como tal, se os fragmentos forem ordenados crescentemente, podemos obter o datagrama ordenado pela ordem em que os dados estavam no datagrama original.

### 1.3.6 Questão f

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

Como podemos verificar pela Figura 1.17, ocorreu, de facto, o processo de fragmentação, onde foram enviados 1480 bytes num primeiro fragmento, 1480 bytes num segundo fragmento e, finalmente, 1121 bytes num terceiro fragmento. Assim sendo, ao somar estes valores com o tamanho do cabeçalho da respetiva *frame* (20 bytes), obtemos um total de 4101 bytes, sendo isto o total de bytes enviados no comando *traceroute* original.

### 1.3.7 Questão g

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

A expressão lógica que permite detetar o último fragmento correspondente ao datagrama original é a seguinte:

```
more_fragments.flag = 0  fragment_offset != 0  datagram.id() == datagram_original.id()
```

## Capítulo 2

# Parte II - Endereçamento e Encaminhamento IP

### Caso de Estudo

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 respetivo 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.

**IMPORTANTE:** Os exercícios 1 e 3 foram feitos no mesmo computador enquanto que o exercício 2 foi feito num computador diferente.

### 2.1 Exercício 1

**Exercício 1.** Atenda aos endereços IP atribuídos automaticamente pelo CORE aos diversos equipamentos.

#### 2.1.1 Questão 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.**

Na figura 2.1, está ilustrada a topologia CORE definida, onde estão descritos os endereços IP e as máscaras de rede (de 24 bits) atribuídos a cada equipamento.

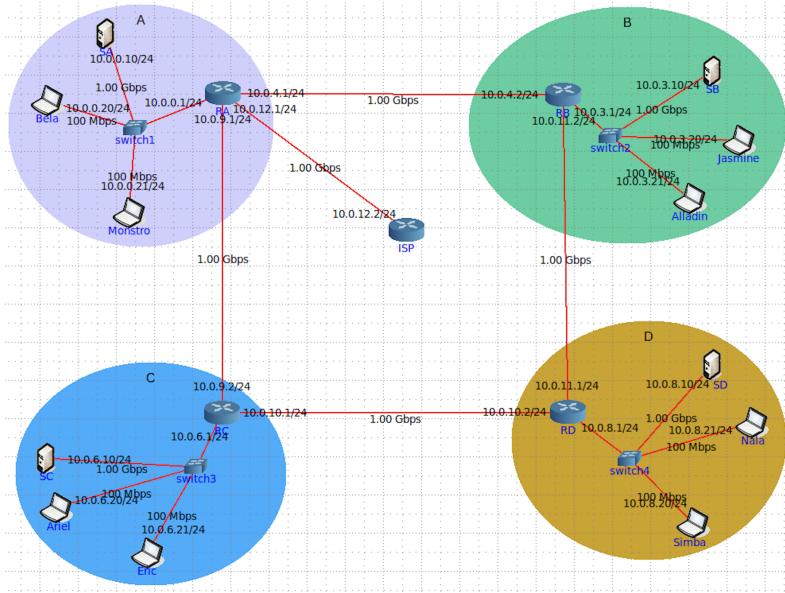


Figura 2.1: Topologia LEI-RC

### 2.1.2 Questão b

**Tratam-se de endereços públicos ou privados? Porquê?**

Tendo em conta que, de acordo com o protocolo RFC 1918, o prefixo 10.0 identifica redes privadas, isto quer dizer que os endereços em questão são privados.

### 2.1.3 Questão c

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

Não é necessário atribuir um endereço IP aos *switches* porque estes encontram-se numa camada mais abaixo que os routers, na camada de ligação de dados (*Link Layer*).

### 2.1.4 Questão d

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

Conforme ilustrado na Figura 2.2, e recorrendo ao comando `ping`, conseguimos perceber que existe conectividade IP dentro de cada departamento.

```

root@Monstro:/tmp/pycore.42223/Monstro.conf# ping 10.0.0.10 -c 4
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=64 time=0.409 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=0.280 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=0.282 ms
64 bytes from 10.0.0.10: icmp_seq=4 ttl=64 time=0.279 ms

--- 10.0.0.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3029ms
rtt min/avg/max/mdev = 0.279/0.312/0.409/0.055 ms
root@Monstro:/tmp/pycore.42223/Monstro.conf# 

root@Ariel:/tmp/pycore.42223/Ariel.conf# ping 10.0.6.10 -c 4
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=64 time=0.388 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=64 time=0.323 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=64 time=0.333 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=64 time=0.288 ms

--- 10.0.6.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
rtt min/avg/max/mdev = 0.288/0.333/0.388/0.035 ms
root@Ariel:/tmp/pycore.42223/Ariel.conf# 

root@Simba:/tmp/pycore.42223/Simba.conf# ping 10.0.8.10 -c 4
PING 10.0.8.10 (10.0.8.10) 56(84) bytes of data.
64 bytes from 10.0.8.10: icmp_seq=1 ttl=64 time=0.386 ms
64 bytes from 10.0.8.10: icmp_seq=2 ttl=64 time=0.344 ms
64 bytes from 10.0.8.10: icmp_seq=3 ttl=64 time=0.250 ms
64 bytes from 10.0.8.10: icmp_seq=4 ttl=64 time=0.502 ms

--- 10.0.8.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.250/0.370/0.502/0.050 ms
root@Simba:/tmp/pycore.42223/Simba.conf# 

root@Alladin:/tmp/pycore.42223/Alladin.conf# ping 10.0.3.10 -c 4
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_seq=1 ttl=64 time=0.395 ms
64 bytes from 10.0.3.10: icmp_seq=2 ttl=64 time=0.279 ms
64 bytes from 10.0.3.10: icmp_seq=3 ttl=64 time=0.283 ms
64 bytes from 10.0.3.10: icmp_seq=4 ttl=64 time=0.286 ms

--- 10.0.3.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3068ms
rtt min/avg/max/mdev = 0.279/0.308/0.395/0.044 ms
root@Alladin:/tmp/pycore.42223/Alladin.conf# 

```

Figura 2.2: Teste à conectividade IP interna de cada departamento.

### 2.1.5 Questão e

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

```

root@Eric:/tmp/pycore.42223/Eric.conf# ping 10.0.0.10 -c 4
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=1.13 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=62 time=0.672 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=62 time=0.599 ms
64 bytes from 10.0.0.10: icmp_seq=4 ttl=62 time=0.923 ms

--- 10.0.0.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3050ms
rtt min/avg/max/mdev = 0.599/0.830/1.128/0.203 ms
root@Eric:/tmp/pycore.42223/Eric.conf#
root@Eric:/tmp/pycore.42223/Eric.conf#
root@Eric:/tmp/pycore.42223/Eric.conf#
root@Eric:/tmp/pycore.42223/Eric.conf# ping 10.0.3.10 -c 4
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_seq=1 ttl=61 time=1.10 ms
64 bytes from 10.0.3.10: icmp_seq=2 ttl=61 time=0.785 ms
64 bytes from 10.0.3.10: icmp_seq=3 ttl=61 time=0.804 ms
64 bytes from 10.0.3.10: icmp_seq=4 ttl=61 time=0.879 ms

--- 10.0.3.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 0.785/0.881/1.099/0.124 ms
root@Eric:/tmp/pycore.42223/Eric.conf#
root@Eric:/tmp/pycore.42223/Eric.conf#
root@Eric:/tmp/pycore.42223/Eric.conf# ping 10.0.8.10 -c 4
PING 10.0.8.10 (10.0.8.10) 56(84) bytes of data.
64 bytes from 10.0.8.10: icmp_seq=1 ttl=62 time=0.681 ms
64 bytes from 10.0.8.10: icmp_seq=2 ttl=62 time=0.795 ms
64 bytes from 10.0.8.10: icmp_seq=3 ttl=62 time=0.741 ms
64 bytes from 10.0.8.10: icmp_seq=4 ttl=62 time=0.906 ms

--- 10.0.8.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3067ms
rtt min/avg/max/mdev = 0.681/0.771/0.908/0.063 ms
root@Eric:/tmp/pycore.42223/Eric.conf# 

```

Figura 2.3: Teste IP entre os varios Departamentos.

### 2.1.6 Questão f

Verifique se existe conectividade IP do portátil Bela para o router de acesso  $R_{ISP}$ .

Tal como se confirma pela Figura 2.4, existe conexão entre o portátil Bela e o router  $R_{ISP}$ .

```
root@Bela:/tmp/pycore.42223/Bela.conf# ping 10.0.12.2 -c 6
PING 10.0.12.2 (10.0.12.2) 56(84) bytes of data.
64 bytes from 10.0.12.2: icmp_seq=1 ttl=63 time=0.589 ms
64 bytes from 10.0.12.2: icmp_seq=2 ttl=63 time=1.77 ms
64 bytes from 10.0.12.2: icmp_seq=3 ttl=63 time=0.446 ms
64 bytes from 10.0.12.2: icmp_seq=4 ttl=63 time=1.80 ms
64 bytes from 10.0.12.2: icmp_seq=5 ttl=63 time=0.474 ms
64 bytes from 10.0.12.2: icmp_seq=6 ttl=63 time=1.94 ms

--- 10.0.12.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5038ms
rtt min/avg/max/mdev = 0.446/1.170/1.936/0.670 ms
root@Bela:/tmp/pycore.42223/Bela.conf# []
```

Figura 2.4: Teste IP entre a Bela e o  $R_{ISP}$ .

## 2.2 Exercício 2

**Exercício 2.** Para o router RA e o portátil Bela:

### 2.2.1 Questão 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`).

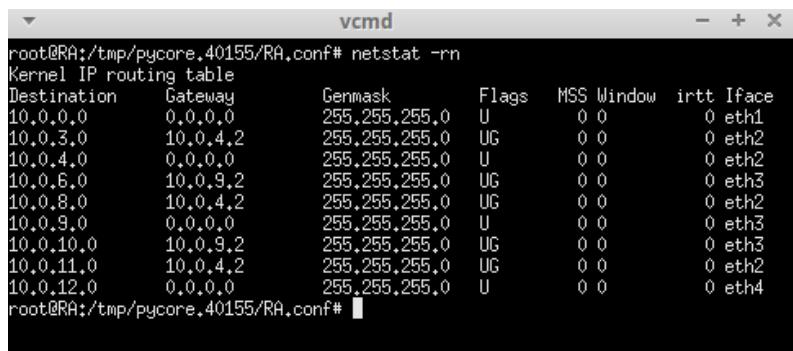
Nas figuras 2.5 e 2.6, temos as tabelas de encaminhamento do router RA e do portátil Bela, respetivamente.

Os campos **Destination** e **Genmask** descrevem o endereço IP destino e a máscara de rede utilizada, respetivamente. A título de exemplo, na segunda linha da tabela de encaminhamento do router RA, o endereço IP destino é 10.0.3.0 e a máscara é 255.255.255.0, o que quer dizer que são utilizados 24 bits para a máscara de rede. Ambos podem ser escritos como 10.0.3.0/24.

O campo **Getaway** contém informação acerca do próximo salto. Por outras palavras, este aponta para o endereço IP através do qual a rede identifica pelo IP destino pode ser acessada.

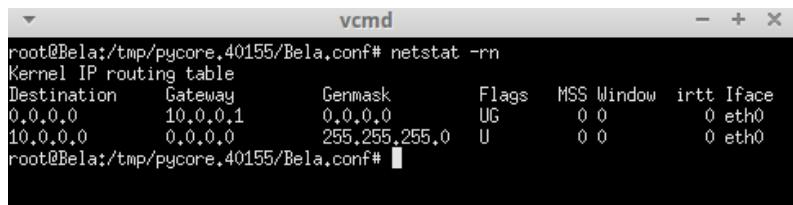
O campo **Flags** permite saber se existe uma rota direta para o endereço IP destino (U) ou, se para chegar ao mesmo, tem de ser tomado o próximo salto (UG).

Finalmente, o campo **irtt Iface** destina-se à identificação da interface de saída da máquina local, responsável por alcançar o getaway.



Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
10.0.0.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
10.0.3.0	10.0.4.2	255.255.255.0	UG	0	0	0	eth2
10.0.4.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2
10.0.6.0	10.0.9.2	255.255.255.0	UG	0	0	0	eth3
10.0.8.0	10.0.4.2	255.255.255.0	UG	0	0	0	eth2
10.0.9.0	0.0.0.0	255.255.255.0	U	0	0	0	eth3
10.0.10.0	10.0.9.2	255.255.255.0	UG	0	0	0	eth3
10.0.11.0	10.0.4.2	255.255.255.0	UG	0	0	0	eth2
10.0.12.0	0.0.0.0	255.255.255.0	U	0	0	0	eth4

Figura 2.5: Tabela de encaminhamento do router RA



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

Figura 2.6: Tabela de encaminhamento do portátil Bela

### 2.2.2 Questão 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).

A partir da Figura 2.7, reparamos que está a ser executado o protocolo OSPF6, um protocolo de encaminhamento dinâmico, que é responsável em encaminhar os pacotes de rede pelo melhor caminho possível. Assim sendo, concluimos que está a ser usado encaminhamento dinâmico.

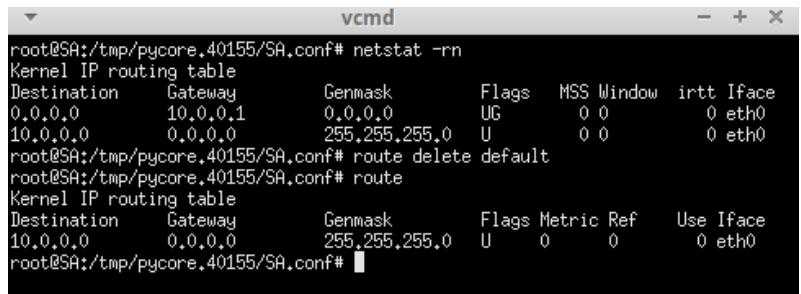
```
root@RA:/tmp/pycore.40155/RA.conf# ps -ax
  PID TTY      STAT      TIME COMMAND
    1 ?        S          0:00 vnodesd -v -c /tmp/pycore.40155/RA -l /tmp/pycore.40
    67 ?       Ss         0:00 /usr/local/sbin/zebra -d
    74 ?       Ss         0:00 /usr/local/sbin/ospf6d -d
    78 ?       Ss         0:00 /usr/local/sbin/ospfd -d
   92 pts/2    Ss         0:00 /bin/bash
  100 pts/2    R+        0:00 ps -ax
root@RA:/tmp/pycore.40155/RA.conf#
```

Figura 2.7: Tabela de encaminhamento do portátil Bela da rede A

### 2.2.3 Questão 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.

A rota por defeito é utilizada sempre que não existe uma entrada específica na tabela para a rede destino, tendo menos prioridade que as outras rotas. Deste modo, ao remover a rota `default` do servidor SA (Figura 2.8), este apenas encaminhará pacotes para a sua rede local (10.0.0.0). Isto implicará a impossibilidade de comunicar com subredes com outro endereço IP. Logo, todos os pacotes direcionados para endereços IP que não pertençam à sua rede local serão recebidos, mas não respondidos.



```
root@SA:/tmp/pycore.40155/SA.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@SA:/tmp/pycore.40155/SA.conf# route delete default
root@SA:/tmp/pycore.40155/SA.conf# route
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
10.0.0.0        0.0.0.0        255.255.255.0  U        0 0          0 eth0
root@SA:/tmp/pycore.40155/SA.conf#
```

Figura 2.8: Execução do comando `route delete default` para o servidor SA

## 2.2.4 Questão 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.**

Na figura 2.9, estão a ser adicionadas as rotas estáticas ao servidor SA. No entanto, como um dos comandos se encontra encurtado, apresentamos também aqui os 4 comandos feitos, na mesma ordem que aparecem na figura:

```
route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.0.1
route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.0.1
route add 10.0.0.0 gw 10.0.0.1
route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.0.1
```

```
<A.conf# route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.0.1
<A.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.0.1
root@SA:/tmp/pycore.40155/SA.conf# route add 10.0.0.0 gw 10.0.0.1
root@SA:/tmp/pycore.40155/SA.conf# route add -net 10.0.6.0 netmask 255.255.255.0
root@SA:/tmp/pycore.40155/SA.conf#
```

Figura 2.9: Adição das rotas estáticas ao servidor SA

## 2.2.5 Questão 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.**

Tal como podemos observar pelas Figuras 2.10, 2.11 e 2.13, o servidor já consegue comunicar com subredes de endereço IP diferente do da sua rede local. Na figura 2.12, temos um teste à conectividade com um endereço IP da sua rede local.

```
root@SA:/tmp/pycore.40155/SA.conf# ping 10.0.8.10
PING 10.0.8.10 (10.0.8.10) 56(84) bytes of data.
64 bytes from 10.0.8.10: icmp_seq=1 ttl=61 time=2.48 ms
64 bytes from 10.0.8.10: icmp_seq=2 ttl=61 time=1.16 ms
64 bytes from 10.0.8.10: icmp_seq=3 ttl=61 time=1.75 ms
64 bytes from 10.0.8.10: icmp_seq=4 ttl=61 time=1.32 ms
64 bytes from 10.0.8.10: icmp_seq=5 ttl=61 time=1.82 ms
```

Figura 2.10: ping 10.0.8.10

```
root@SA:/tmp/pycore.40155/SA.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_seq=1 ttl=62 time=1.50 ms
64 bytes from 10.0.3.10: icmp_seq=2 ttl=62 time=0.859 ms
64 bytes from 10.0.3.10: icmp_seq=3 ttl=62 time=2.65 ms
64 bytes from 10.0.3.10: icmp_seq=4 ttl=62 time=3.42 ms
64 bytes from 10.0.3.10: icmp_seq=5 ttl=62 time=0.858 ms
```

Figura 2.11: ping 10.0.3.10

```
root@SA:/tmp/pycore.40155/SA.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=64 time=0.029 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=0.041 ms
64 bytes from 10.0.0.10: icmp_seq=4 ttl=64 time=0.025 ms
64 bytes from 10.0.0.10: icmp_seq=5 ttl=64 time=0.043 ms
```

Figura 2.12: ping 10.0.0.10

```
root@SA:/tmp/pycore.40155/SA.conf# ping 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=62 time=2.77 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=62 time=0.839 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=62 time=1.47 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=62 time=0.946 ms
64 bytes from 10.0.6.10: icmp_seq=5 ttl=62 time=0.963 ms
```

Figura 2.13: ping 10.0.6.10

Por fim, na figura 2.14, temos a nova tabela de encaminhamento do servidor SA, na qual se encontram as rotas estáticas adicionadas anteriormente.

```
root@SA:/tmp/pycore.40155/SA.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt I
face
10.0.0.0       10.0.0.1      255.255.255.255 UGH        0 0        0 e
th0
10.0.0.0       0.0.0.0       255.255.255.0   U          0 0        0 e
th0
10.0.3.0       10.0.0.1      255.255.255.0   UG         0 0        0 e
th0
10.0.6.0       10.0.0.1      255.255.255.0   UG         0 0        0 e
th0
10.0.8.0       10.0.0.1      255.255.255.0   UG         0 0        0 e
th0
```

Figura 2.14: Tabela de encaminhamento do servidor SA

## 2.3 Definição de Sub-redes

**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.

### 2.3.1 Questão 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 subredes são usáveis. Justifique as opções tomadas no planeamento.

Na figura 2.15, está ilustrado o esquema de endereçamento para as redes dos departamentos. O endereço de rede IP é 192.168.101.128 e, convertendo para binário, fica 11000000.10101000.01100101.10000000.

Como é necessário 4 sub-redes, foram utilizados 3 bits para a sub-rede e, deste modo, ainda restam outras para possíveis expansões futuras. Deste modo, sobram 4 bits para o host. Desses 4 bits, o número de possibilidades é  $2^4 - 2$ , onde é retirada a probabilidade do host ser 0000 ou 1111.

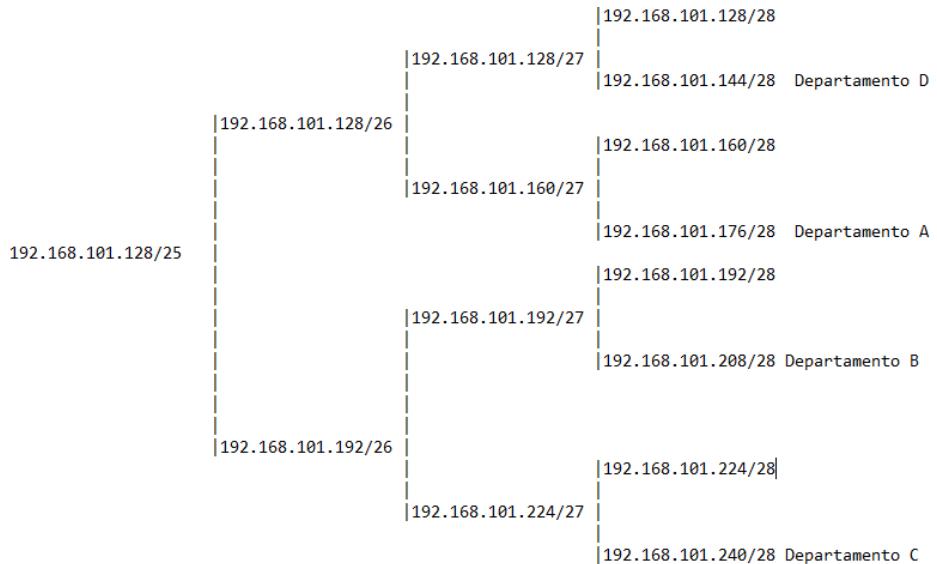


Figura 2.15: Esquema de endereçamento

### 2.3.2 Questão 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 utilizada foi 28. Em cada departamento, podemos interligar 14 ( $24 - 2 = 14$ ) hosts IP. Temos, também, 4 ( $23 - 4 = 4$ ) prefixos de sub-rede disponíveis para o futuro.

### 2.3.3 Questão 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, em cada uma das sub-redes, modificou-se os endereços IP e as suas respectivas máscaras de rede manualmente, tal como se pode visualizar na Figura 2.16. Seguidamente, com o intuito de averiguar se a tabela de endereçamento apresentava caminhos para todos os departamentos, foi utilizado o comando `netstat -rn`. Finalmente, para testar o estabelecimento da ligação, recorreu-se ao comando `ping`, executado num *host* de cada departamento.

The figure consists of four vertically stacked terminal windows, each titled "vcmd".

- Terminal 1 (top):** Shows the command `root@S0A:/tmp/pycore_36029/S0.conf# netstat -rn`. The output displays the kernel IP routing table with one route to department A (192.168.101.177).
- Terminal 2:** Shows the command `root@S0B:/tmp/pycore_36029/S0.conf# ping 192.168.101.244 -c 5`. The output shows 5 packets transmitted, 5 received, 0% packet loss, and a round-trip time of 4073ms.
- Terminal 3:** Shows the command `root@S0B:/tmp/pycore_36029/S0.conf# netstat -rn`. The output displays the kernel IP routing table with one route to department B (192.168.101.176).
- Terminal 4:** Shows the command `root@S0B:/tmp/pycore_36029/S0.conf# ping 192.168.101.180 -c 5`. The output shows 5 packets transmitted, 5 received, 0% packet loss, and a round-trip time of 4070ms.
- Terminal 5:** Shows the command `root@S0C:/tmp/pycore_36029/S0.conf# netstat -rn`. The output displays the kernel IP routing table with one route to department C (192.168.101.209).
- Terminal 6:** Shows the command `root@S0C:/tmp/pycore_36029/S0.conf# ping 192.168.101.190 -c 5`. The output shows 5 packets transmitted, 5 received, 0% packet loss, and a round-trip time of 4008ms.
- Terminal 7:** Shows the command `root@S0D:/tmp/pycore_36029/S0.conf# netstat -rn`. The output displays the kernel IP routing table with one route to department D (192.168.101.240).
- Terminal 8:** Shows the command `root@S0D:/tmp/pycore_36029/S0.conf# ping 192.168.101.210 -c 5`. The output shows 5 packets transmitted, 5 received, 0% packet loss, and a round-trip time of 4083ms.

Figura 2.16: Teste à conectividade IP interna na rede local LEI-RC

# Capítulo 3

## Conclusão

Com a realização deste trabalho prático, foi possível solidar a matéria lecionada nas aulas teóricas acerca dos temas **Datagramas IP e Fragmentação** e **Endereçamento e Encaminhamento IP**, bem como conhecer as ferramentas *CORE* e *Wireshark*, usadas para gestão e análise de tráfego de redes.

No âmbito do primeiro tema mencionado, relativo à primeira parte deste trabalho prático – **Datagramas IP e Fragmentação** –, preparou-se uma topologia *CORE* de maneira a averiguar o comportamento do comando `traceroute`, o que nos permitiu saber mais sobre a estrutura dos datagramas IP. Para além disto, ao analisar os campos de dados do datagrama recebido, pode-se deduzir se é necessário fragmentar o pacote inicial caso o tamanho do mesmo ultrapasse o valor de MTU. Ainda com base no cabeçalho do datagrama, mais precisamente através da verificação das *flags*, pode-se inferir se o datagrama foi fragmentado, e em caso afirmativo, se um determinado fragmento é o primeiro ou se existem mais.

No âmbito do segundo tema mencionado, relativo à segunda parte deste trabalho prático – **Endereçamento e Encaminhamento IP** –, preparou-se uma nova topologia mais complexa, organizada em 4 departamentos, cada um associado a uma sub-rede (A, B, C, D). A resolução desta segunda parte, para além de solidar o que já sabíamos sobre os mecanismos de *subnetting* e sobre a aplicação de máscaras de rede, permitiu consultar uma tabela de encaminhamento *unicast* e saber interpretar cada uma das suas entradas. Com base nisto, foi, ainda, possível verificar as implicações da remoção da rota por defeito e testar a conetividade IP interna a cada departamento através do comando `ping`.