

TP2: Protocolo IPv4

Ana Silva (a91678), Paulo Freitas (a100053) e Rúben Machado (a91656)

Questões e Respostas

(Parte I)

1 Datagramas IP e Fragmentação

1.1 Captura de tráfego IP

- Para verificar o comportamento do *traceroute*, implemente no CORE uma topologia retangular com um *router* em cada vértice. Ligue a cada um dos *routers* um *host* (pc) e atribua à rede de cada *host* os endereços 192.<nº do grupo+N>.<nº do grupo+N>.X/24, com N=0,1,2,3. Atribua ao decimal X um valor adequado. Atribua a este *host* o nome PC1.

Ao *host* que está ligado ao *router* diametralmente oposto do PC1 atribua o nome PC2.

Coloque esses nomes nos respectivos *hosts* da topologia e arranque a rede. Ative o *Wireshark* no *host* PC1. Numa *shell* do PC1, execute o comando *traceroute -I* para o endereço IP do PC2.

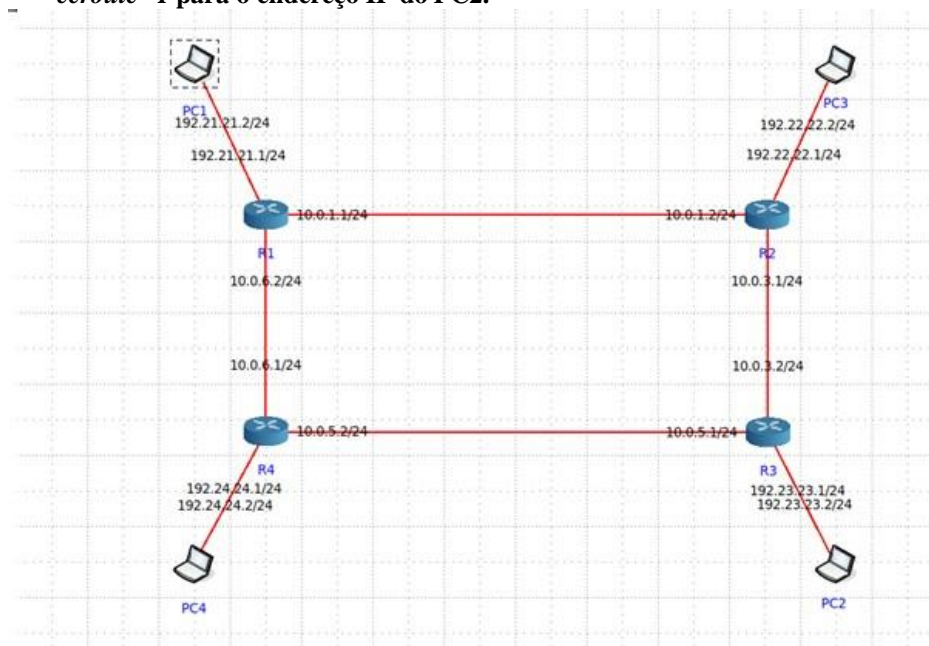


Fig. 1. Topologia do exercício 1

- a) Registe e analise o tráfego ICMP enviado pelo PC1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

As tramas enviadas pelo PC1 são divididas em 3 *probes* de modo a identificar quantos “saltos” são necessários para comunicar com o PC2 (Fig.2).

As primeiras tramas enviadas pelo PC1 não obtêm respostas, porque o TTL (tempo de vida) não é suficiente para alcançar o PC2. Quando o TTL chega com sucesso ao *host* destino, então o segundo *host* envia uma trama ICMP com TTL 61 (64 corresponde ao valor *default*), pois não sabe a que distância se encontra do PC1.

```
root@PC1:/tmp/pycore.34041/PC1.conf# traceroute -I 192.23.23.2
traceroute to 192.23.23.2 (192.23.23.2), 30 hops max, 60 byte packets
 1 192.21.21.1 (192.21.21.1) 0.040 ms 0.004 ms 0.003 ms
 2 10.0.1.2 (10.0.1.2) 0.011 ms 0.005 ms 0.005 ms
 3 10.0.3.2 (10.0.3.2) 0.021 ms 0.006 ms 0.005 ms
 4 192.23.23.2 (192.23.23.2) 0.018 ms 0.007 ms 0.007 ms
root@PC1:/tmp/pycore.34041/PC1.conf#
```

Fig. 2. Tráfego ICMP

- b) Qual deve ser o valor inicial mínimo do campo TTL para alcançar o PC2? Verifique na prática que a sua resposta está correta.

O valor inicial do campo TTL para alcançar o PC2 corresponde a 4. No qual através do tráfego ICMP podemos verificar que na prática o valor esperado é correto (Fig.3).

No.	Time	Source	Destination	Protocol	Length	Info
11	6.342597771	192.21.21.1	192.21.21.2	ICMP	74	Echo (ping) request id=0x0022, seq=1/254, ttl=1 (no response found)
12	6.342722346	192.21.21.1	192.21.21.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
13	6.342777134	192.21.21.1	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=2/512, ttl=1 (no response found)
14	6.342811681	192.21.21.1	192.23.23.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
15	6.342811725	192.21.21.1	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=3/1536, ttl=1 (no response found)
16	6.342833695	192.21.21.1	192.21.21.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
17	6.342850787	192.21.21.1	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=4/3072, ttl=2 (no response found)
18	6.342850821	192.21.21.1	192.23.23.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
19	6.342897629	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=5/1280, ttl=2 (no response found)
20	6.342891165	10.0.1.2	192.21.21.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
21	6.342895335	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=6/1536, ttl=2 (no response found)
22	6.342898727	10.0.1.2	192.21.21.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
23	6.342898855	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=7/3192, ttl=3 (no response found)
24	6.342898905	10.0.1.2	192.21.21.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
25	6.342930417	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=8/2048, ttl=3 (no response found)
26	6.342935140	10.0.1.2	192.21.21.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
27	6.342931695	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=9/2208, ttl=3 (no response found)
28	6.342941952	10.0.1.2	192.21.21.2	ICMP	102	time-to-live exceeded (time to live exceeded in transit)
29	6.342944046	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=10/2568, ttl=4 (reply in 30)
30	6.342946297	192.21.21.2	192.21.21.2	ICMP	74	Echo (ping) reply id=0x0022, seq=10/2568, ttl=61 (request in 29)
31	6.342946522	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=11/2816, ttl=4 (reply in 32)
32	6.3429471421	192.21.21.2	192.21.21.2	ICMP	74	Echo (ping) reply id=0x0022, seq=11/2816, ttl=61 (request in 31)
33	6.342947287	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=12/3072, ttl=4 (reply in 34)
34	6.3429479118	192.21.21.2	192.21.21.2	ICMP	74	Echo (ping) reply id=0x0022, seq=12/3072, ttl=61 (request in 33)
35	6.3429481240	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=13/3328, ttl=5 (reply in 36)
36	6.3429487872	192.21.21.2	192.21.21.2	ICMP	74	Echo (ping) reply id=0x0022, seq=13/3328, ttl=61 (request in 35)
37	6.3429488676	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=14/3584, ttl=5 (reply in 38)
38	6.3429494662	192.21.21.2	192.21.21.2	ICMP	74	Echo (ping) reply id=0x0022, seq=14/3584, ttl=61 (request in 37)
39	6.3429496419	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=15/3840, ttl=5 (reply in 40)
40	6.3429496983	192.21.21.2	192.21.21.2	ICMP	74	Echo (ping) reply id=0x0022, seq=15/3840, ttl=61 (request in 39)
41	6.3429504068	192.21.21.2	192.23.23.2	ICMP	74	Echo (ping) request id=0x0022, seq=16/4096, ttl=6 (reply in 42)
42	6.3429510186	192.21.21.2	192.21.21.2	ICMP	74	Echo (ping) reply id=0x0022, seq=16/4096, ttl=61 (request in 41)

Fig. 3. Mensagens ICMP

- c) Calcule o valor médio do tempo de ida-e-volta (*Round-Trip Time*) obtido?

O tempo médio é 0.0067 ms, correspondente à última linha (linha 4), no qual tem representado os tempos dos 10 *probes* de envio e resposta (Fig.2).

- 2) Pretende-se agora usar o *tracert* na sua máquina nativa, disponibilizado no Windows não permite mudar o tamanho das mensagens a enviar. Porém, no *Linux/Unix*, o *tracert* permite indicar o tamanho do pacote ICMP através da linha de comando, a seguir ao *host* de destino. Através da análise do cabeçalho IP diga:

```
Microsoft Windows [Version 10.0.22621.819]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\ruben>tracert marco.uminho.pt

Tracing route to marco.uminho.pt [193.136.9.240]
over a maximum of 30 hops:

  1      1 ms      2 ms      1 ms    172.26.254.254
  2      1 ms      1 ms      1 ms    172.16.2.1
  3      2 ms      1 ms      1 ms    172.16.115.252
  4      2 ms      2 ms      2 ms    marco.uminho.pt [193.136.9.240]

Trace complete.
```

Fig. 4. Traceroute em marco.uminho.pt

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

O endereço IP da interface ativa do computador é 172.26.42.81 (Fig.5 - vermelho).

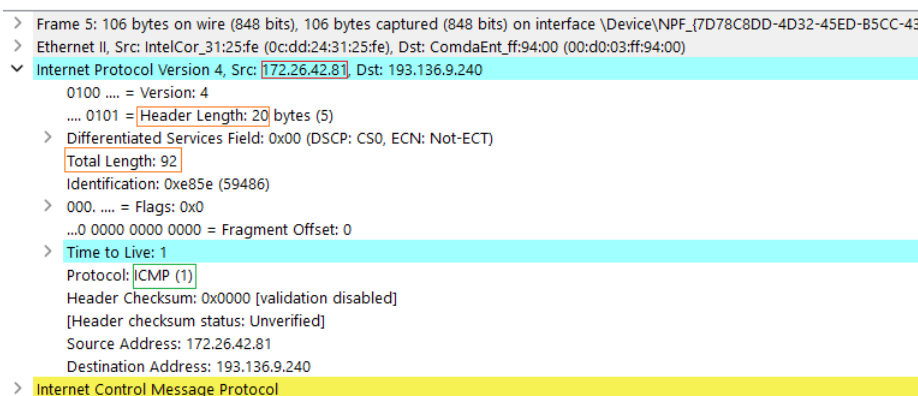


Fig. 5. Campo IPv4

- b) Qual é o valor do campo protocolo? O que identifica?

O valor do campo protocolo é 1 (Fig5 - verde), no qual identifica o *Internet Protocol* (IP).

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

O cabeçalho IPv4 tem 20 bytes, nos quais são o *payload* do datagrama é calculado através da subtração do *total length* e *header length*, ou seja, entre 92 e 20 respetivamente. Esta operação indica que o tamanho do *payload* é 72 bytes (Fig 5 – laranja).

4

d) O datagrama IP foi fragmentado? Justifique.

Como o *Fragment Offset* tem como valor 0, ou seja, o pacote não se encontra fragmentado no início, e o *More Fragments* não se encontra definido, ou seja, não se encontra fragmentado no final. Logo, o datagrama IP não foi fragmentado (Fig.6).

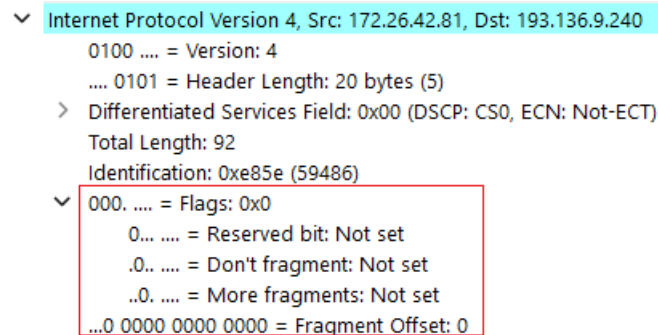


Fig. 6. *Flags* no campo IPv4

e) Ordene os pacotes capturados de acordo com o endereço IP fonte, 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.

A máquina está a tentar comunicar com o router (Fig.7), com o endereço IP de destino (193.136.6.240). Na teoria, será necessário que o TTL seja chamado por 4 vezes. Contudo, tal não se vai concretizar, visto que, inicialmente, a máquina não consegue encontrar o seu destinatário, no que resulta no envio das mensagens ICMP *TTL exceeded*. Eventualmente, a máquina encontrará o seu percurso chegando assim ao destino, com o menor número de TTLs (4). Nos casos onde o TTL é bastante superior ao número de TTLs (61) indica para que a comunicação seja garantida vai ser utilizada o maior número de TTLs possíveis.

No.	Time	Source	Destination	Protoc	Length	BSS Id	Info
5	0.010164	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=25/6400, ttl=1 (no response found)
6	0.011711	172.26.254.254	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
7	0.012404	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=26/6656, ttl=1 (no response found)
8	0.013863	172.26.254.254	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
9	0.014420	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=27/6912, ttl=1 (no response found)
10	0.015986	172.26.254.254	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
43	5.969968	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=28/7168, ttl=2 (no response found)
44	5.971489	172.16.2.1	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
45	5.972592	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=29/7424, ttl=2 (no response found)
46	5.973967	172.16.2.1	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
47	5.974724	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=30/7680, ttl=2 (no response found)
48	5.976028	172.16.2.1	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
61	11.963692	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=31/7936, ttl=3 (no response found)
62	11.965409	172.16.115.252	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
63	11.966485	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=32/8192, ttl=3 (no response found)
64	11.968008	172.16.115.252	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
65	11.968571	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=33/8448, ttl=3 (no response found)
66	11.970323	172.16.115.252	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
79	17.933136	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=34/8704, ttl=4 (reply in 80)
80	17.936325	193.136.9.240	172.26.42.81	ICMP	106		Echo (ping) reply id=0x0001, seq=34/8704, ttl=61 (request in 79)

Fig. 7. Mensagens ICMP e TTL *exceeded*

- f) Indique o padrão observado nos valores do campo de Identificação do datagrama IP e TTL.

O valor do campo de Identificação do datagrama IP em que este vai ser constante, ao longo do tempo. No caso do TTL, o valor vai sendo incrementado sequencialmente.

- 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?

O valor do campo TTL nas respostas ICMP TTL *exceeded* tem valor constante igual a 1 (Fig.8). Isso deve-se ao facto de o *host* de origem não sabe a quantidade de TTL necessários para chegar ao *host* de destino. Portanto, o *host* de origem envia 3 *probes request* com o valor mínimo possível, ou seja, com o TTL igual a 1, caso não seja suficiente é lhe enviada a mensagem ICMP *exceeded*, de seguida vai executar o mesmo processo, mas para o valor TTL igual a 2, assim sucessivamente até chegar ao *host* de destino. Por outro lado, o *host* de destino não sabe a localização do *host* de origem, e, portanto, vai utilizar o número máximo de TTLS (64 por *default*) para garantir que a mensagem chegue ao *host* de origem.

No.	Time	Source	Destination	Protoc	Length	BSS Id	Info
5	0.010164	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=25/6400, ttl=1 (no response found)
6	0.011711	172.26.42.254	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
7	0.012404	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=26/6656, ttl=1 (no response found)
8	0.013863	172.26.42.254	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
9	0.014420	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=27/6912, ttl=1 (no response found)
10	0.015886	172.26.42.254	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
43	5.969968	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=28/7168, ttl=2 (no response found)
44	5.971459	172.26.42.81	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
45	5.972592	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=29/7424, ttl=2 (no response found)
46	5.973967	172.16.2.1	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
47	5.974724	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=30/7680, ttl=2 (no response found)
48	5.976028	172.16.2.1	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
61	11.963692	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=31/7936, ttl=3 (no response found)
62	11.965409	172.16.115.232	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
63	11.966485	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=32/8192, ttl=3 (no response found)
64	11.968093	172.16.115.232	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
65	11.968571	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=33/8448, ttl=3 (no response found)
66	11.970323	172.16.115.232	172.26.42.81	ICMP	70		Time-to-live exceeded (Time to live exceeded in transit)
79	17.933136	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=34/8704, ttl=4 (reply in 80)
80	17.936325	193.136.9.240	172.26.42.81	ICMP	106		Echo (ping) reply id=0x0001, seq=34/8704, ttl=61 (request in 79)
81	17.937916	172.26.42.81	193.136.9.240	ICMP	106		Echo (ping) request id=0x0001, seq=35/8960, ttl=4 (reply in 82)
83	17.941954	193.136.9.240	172.26.42.81	ICMP	106		Echo (ping) reply id=0x0001, seq=35/8960, ttl=61 (request in 81)

<p>[Checksum Status: Good]</p> <p>Unused: 00000000</p> <p>Internet Protocol Version 4, Src: 172.26.42.81, Dst: 193.136.9.240</p> <p>0100 ... = Version: 4</p> <p>... 0101 = Header Length: 20 bytes (5)</p> <p>Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)</p> <p>Total Length: 92</p> <p>Identification: 0xe85e (59486)</p> <p>000 ... = Flags: 0x0</p> <p>... 0000 0000 0000 = Fragment Offset: 0</p> <p>Time to Live: 1</p> <p>Protocol: ICMP (1)</p> <p>Header Checksum: 0x2f5f (validation disabled)</p> <p>[Header checksum status: Unverified]</p> <p>Source Address: 172.26.42.81</p> <p>Destination Address: 193.136.9.240</p> <p>Internet Control Message Protocol</p> <p>Type: 8 (Echo (ping) request)</p> <p>Code: 0</p> <p>Checksum: 0x7fe5 (unverified) [in ICMP error packet]</p> <p>[Checksum Status: Unverified]</p>	<pre> 0000 0c dd 24 31 25 fe 00 00 03 ff 94 00 08 00 45 c0 ..\$1%.....E 0010 00 38 7e b7 00 00 ff 01 88 2c 1a fe fe ac 1a 8.....E..^ 0020 2a 51 0b 00 f4 ff 00 00 00 00 45 00 00 5c e8 5e *Q.....E..^ 0030 00 00 01 01 2f 5f ac 1a 2a 51 c1 88 09 f0 08 00 /...*Q..... 0040 f7 e5 00 01 00 19 </pre>
---	--

Fig. 8. TTL de uma mensagem ICMP TTL *exceeded*

- 3) Pretende-se agora analisar a fragmentação de pacotes IP. Capture com o *Wireshark* o tráfego gerado pelo comando `ping <opção> 5221 marco.uminho.pt`, onde a opção `-l` (*Windows*) ou `-s` (*Linux*) define o número de bytes enviados no campo de dados do pacote ICMP. Observe o tráfego capturado.

```
C:\Users\ruben>ping -l 5221 marco.uminho.pt

Pinging marco.uminho.pt [193.136.9.240] with 5221 bytes of data:
Reply from 193.136.9.240: bytes=5221 time=3ms TTL=61
Reply from 193.136.9.240: bytes=5221 time=3ms TTL=61
Reply from 193.136.9.240: bytes=5221 time=3ms TTL=61
Reply from 193.136.9.240: bytes=5221 time=3ms TTL=61

Ping statistics for 193.136.9.240:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 3ms, Maximum = 3ms, Average = 3ms
```

Fig. 9. Ping para marco.uminho.pt

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

A primeira mensagem ICMP foi fragmenta (Fig.10), visto que o seu tamanho é superior ao que à capacidade de informação da rede (MTU), ou seja, o router recorre à fragmentação do pacote para garantir que não haja perda de informação.

No.	Time	Source	Destination	Protoc	Lengt	BSS Id	Info
*	1 0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e86e) [Reassembled in #4]
*	2 0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e86e) [Reassembled in #4]
*	3 0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e86e) [Reassembled in #4]
4	4 0.000000	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=41/10496, ttl=128 (reply in 8)
5	5 0.003558	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=5e5f) [Reassembled in #8]
6	6 0.003575	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=5e5f) [Reassembled in #8]
7	7 0.003580	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=5e5f) [Reassembled in #8]
8	8 0.003584	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=41/10496, ttl=61 (request in 4)
9	9 1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e86f) [Reassembled in #12]
10	10 1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e86f) [Reassembled in #12]
11	11 1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e86f) [Reassembled in #12]
12	12 1.009022	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=42/10752, ttl=128 (reply in 16)
13	13 1.012585	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=6192) [Reassembled in #16]
14	14 1.012604	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6192) [Reassembled in #16]
15	15 1.012608	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=6192) [Reassembled in #16]
16	16 1.012612	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=42/10752, ttl=61 (request in 12)
17	17 2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e870) [Reassembled in #20]
18	18 2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e870) [Reassembled in #20]
19	19 2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e870) [Reassembled in #20]
20	20 2.022898	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=43/11008, ttl=128 (reply in 24)
21	21 2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=649d) [Reassembled in #24]
22	22 2.026521	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=649d) [Reassembled in #24]

Fig. 10. Fragmento da mensagem ICMP

- 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?

As *flags* no cabeçalho indicam se o datagrama foi fragmentado, e o *Fragment Offset* indica a posição do fragmento. Portanto, o datagrama (Fig.11) indica que existe mais fragmentos (*More Fragments*), e ainda o *Fragment Offset* tem como valor 0 (Fig.11 – vermelho), logo vai corresponder ao primeiro fragmento, sendo o seu tamanho 1500 bytes (Fig.11 - verde).

No.	Time	Source	Destination	Protoc	Lengt	BSS Id	Info
1	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e86e) [Reassembled in #4]
2	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e86e) [Reassembled in #4]
3	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e86e) [Reassembled in #4]
4	0.000000	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=41/10496, ttl=128 (reply in 8)
5	0.003558	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=5e5f) [Reassembled in #8]
6	0.003575	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=5e5f) [Reassembled in #8]
7	0.003580	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=5e5f) [Reassembled in #8]
8	0.003584	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=41/10496, ttl=61 (request in 4)
9	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e86f) [Reassembled in #12]
10	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e86f) [Reassembled in #12]
11	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e86f) [Reassembled in #12]
12	1.009022	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=42/10752, ttl=128 (reply in 16)
13	1.012585	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=6192) [Reassembled in #16]
14	1.012604	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6192) [Reassembled in #16]
15	1.012608	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=6192) [Reassembled in #16]
16	1.012612	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=42/10752, ttl=61 (request in 12)
17	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e870) [Reassembled in #20]
18	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e870) [Reassembled in #20]
19	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e870) [Reassembled in #20]
20	2.022898	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=43/11008, ttl=128 (reply in 24)
21	2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=649d) [Reassembled in #24]
22	2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=649d) [Reassembled in #24]
23	2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=649d) [Reassembled in #24]

> Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface [Device]NPF_{7D78C8DD-4D32-45ED-E000-000000000000} (00:00:00:00:00:00) > Ethernet II, Src: IntelCor_31:25:f6 (0c:dd:24:31:25:f6), Dst: ComdEnt_ff:94:00 (00:00:03:ff:94:00) > Internet Protocol Version 4, Src: 172.26.42.81, Dst: 193.136.9.240 > ICMP ... = Version: 4 > ... 0101 = Header Length: 20 bytes (5) > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) > Total Length: 1500 > Identification: 0xe86e (59502) > 001... = Flags: 0x1, More fragments > ... 0000 0000 0000 = Fragment Offset: 0 > Time to Live: 128 > Protocol: ICMP (1) > Header Checksum: 0x0000 (validation disabled) > [Header checksum status: Unverified] > Source Address: 172.26.42.81 > Destination Address: 193.136.9.240 > [Reassembled IPv4 in frame: 4] > Data (1480 bytes)	0000 00 00 03 ff 94 00 0c dd 24 31 25 fe 08 00 45 00 \$1% - E 0010 05 dc e8 6e 20 00 80 01 00 00 ac 1a 2a 51 c1 88 n Q 0020 09 f0 08 00 22 c0 00 01 00 29 61 62 63 64 65 66 j abcd 0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmn opqrstuv 0040 77 78 79 7a 7b 7c 7d 7e 7f 80 81 82 83 84 85 86 wxyzabcd efghijklmn 0050 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 ijklmnop qrstuvw 0060 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 yzabcd efghijklmn 0070 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 bcdefghijklmnop 0080 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 rstuvwab cdefghijklmn 0090 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 80 cdefghijklmnopqrstu 00a0 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 defghijklmnopqrstu 00b0 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 69 vwxyzabcd efghijklmn 00c0 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 xyzabcd efghijklmnopqrstu 00d0 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 yzabcd efghijklmnopqrstu 00e0 76 77 78 79 80 81 82 83 84 85 86 67 68 69 6a 6b 6c vwxyzabcd efghijklmn 00f0 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 xyzabcd efghijklmnopqrstu 0100 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 yzabcd efghijklmnopqrstu 0110 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 abcdefghijklmnopqrstu 0120 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 ijklmnopqrstu vwxyzabcd 0130 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 bcdefghijklmnopqrstu 0140 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 cdefghijklmnopqrstu 0150 73 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 vwxyzabcd efghijklmn 0160 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 xyzabcd efghijklmnopqrstu 0170 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 yzabcd efghijklmnopqrstu 0180 75 76 77 78 79 80 81 82 83 84 85 86 67 68 69 6a 6b 6c vwxyzabcd efghijklmn 0190 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 xyzabcd efghijklmnopqrstu
---	---

Fig. 11. Flags do primeiro fragmento

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

A informação do cabeçalho do fragmento do datagrama IP original transmite que se trata do segundo fragmento, visto que as flags indicam que existe mais fragmentos, como também o fragment offset é diferente de 0. Logo, podemos concluir que se trata do segundo fragmento (Fig.12).

No.	Time	Source	Destination	Protoc	Lengt	BSS Id	Info
1	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e86e) [Reassembled in #4]
2	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e86e) [Reassembled in #4]
3	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e86e) [Reassembled in #4]
4	0.000000	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=41/10496, ttl=128 (reply in 8)
5	0.003558	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=5e5f) [Reassembled in #8]
6	0.003575	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=5e5f) [Reassembled in #8]
7	0.003580	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=5e5f) [Reassembled in #8]
8	0.003584	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=41/10496, ttl=61 (request in 4)
9	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e86f) [Reassembled in #12]
10	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e86f) [Reassembled in #12]
11	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e86f) [Reassembled in #12]
12	1.009022	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=42/10752, ttl=128 (reply in 16)
13	1.012585	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=6192) [Reassembled in #16]
14	1.012604	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=6192) [Reassembled in #16]
15	1.012608	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=6192) [Reassembled in #16]
16	1.012612	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=42/10752, ttl=61 (request in 12)
17	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=e870) [Reassembled in #20]
18	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e870) [Reassembled in #20]
19	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=e870) [Reassembled in #20]
20	2.022898	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=43/11008, ttl=128 (reply in 24)
21	2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=0, ID=649d) [Reassembled in #24]
22	2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=1480, ID=649d) [Reassembled in #24]
23	2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, off=2960, ID=649d) [Reassembled in #24]

> Frame 2: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface [Device]NPF_{7D78C8DD-4D32-45ED-E000-000000000000} (00:00:00:00:00:00) > Ethernet II, Src: IntelCor_31:25:f6 (0c:dd:24:31:25:f6), Dst: ComdEnt_ff:94:00 (00:00:03:ff:94:00) > Internet Protocol Version 4, Src: 172.26.42.81, Dst: 193.136.9.240 > ICMP ... = Version: 4 > ... 0101 = Header Length: 20 bytes (5) > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) > Total Length: 1500 > Identification: 0xe86e (59502) > 001... = Flags: 0x1, More fragments > ... 0000 1011 1001 = Fragment Offset: 1480 > Time to Live: 128 > Protocol: ICMP (1) > Header Checksum: 0x0000 (validation disabled) > [Header checksum status: Unverified] > Source Address: 172.26.42.81 > Destination Address: 193.136.9.240 > [Reassembled IPv4 in frame: 4] > Data (1480 bytes)	0000 00 00 03 ff 94 00 0c dd 24 31 25 fe 08 00 45 00 \$1% - E 0010 05 dc e8 6e 20 00 80 01 00 00 ac 1a 2a 51 c1 88 n Q 0020 09 f0 08 00 22 c0 00 01 00 29 61 62 63 64 65 66 j abcd 0030 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ghijklmn opqrstuv 0040 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 ijklmnop qrstuvw 0050 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 abcdefghijklmnopqrstu 0060 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 rstuvwab cdefghijklmn 0070 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 bcdefghijklmnopqrstu 0080 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 cdefghijklmnopqrstu 0090 73 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 vwxyzabcd efghijklmn 00a0 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 xyzabcd efghijklmnopqrstu 00b0 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 yzabcd efghijklmnopqrstu 00c0 75 76 77 78 79 80 81 82 83 84 85 86 67 68 69 6a 6b 6c vwxyzabcd efghijklmn 00d0 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 yzabcd efghijklmnopqrstu 00e0 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 rstuvwab cdefghijklmn 00f0 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 ijklmnopqrstu vwxyzabcd 0100 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 yzabcd efghijklmnopqrstu 0110 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 bcdefghijklmnopqrstu 0120 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 rstuvwab cdefghijklmn 0130 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 bcdefghijklmnopqrstu 0140 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 cdefghijklmnopqrstu 0150 73 74 75 76 77 78 79 80 81 82 83 84 85 86 67 68 vwxyzabcd efghijklmn 0160 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 xyzabcd efghijklmnopqrstu 0170 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 yzabcd efghijklmnopqrstu 0180 75 76 77 78 79 80 81 82 83 84 85 86 67 68 69 6a 6b 6c vwxyzabcd efghijklmn 0190 6e 6f 70 71 72 73 74 75 76 77 78 79 80 81 82 83 yzabcd efghijklmnopqrstu
--	---

Fig. 12. Flags do segundo fragmento

- d) **Quantos fragmentos foram criados a partir do datagrama original? Como se detecta o último fragmento correspondente ao datagrama original? Estabeleça um filtro no Wireshark que permita listar o último fragmento do datagrama IP segmentado.**

O datagrama original foi fragmentado em 4 fragmentos, visto que cada fragmento (à exceção do último) tem o tamanho de 1500 bytes (incluindo os 20 bytes do *Header Length*). Através das *flags* permite ter conhecimento se existe fragmentos adicionais, ou seja, quando as *flags* correspondem a 0 indica que aquele fragmento é o último. No Wireshark, utilizou-se o filtro (Fig.13) para listar os últimos fragmentos do datagrama IP segmentado.

ip.flags.mf == 0						
No.	Time	Source	Destination	Protoc	Length	BSS Id
4	0.000000	172.26.42.81	193.136.9.240	ICMP	823	
8	0.003584	193.136.9.240	172.26.42.81	ICMP	823	
12	1.009022	172.26.42.81	193.136.9.240	ICMP	823	
16	1.012612	193.136.9.240	172.26.42.81	ICMP	823	
20	2.022898	172.26.42.81	193.136.9.240	ICMP	823	
24	2.026553	193.136.9.240	172.26.42.81	ICMP	823	
28	3.028322	172.26.42.81	193.136.9.240	ICMP	823	
32	3.032002	193.136.9.240	172.26.42.81	ICMP	823	

Fig. 13. Filtro que contém os últimos fragmentos das mensagens ICMP

- 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.**

Existem campos no cabeçalho IP dos diferentes fragmentos que se vão alterando, nomeadamente o deslocamento e a soma de verificação (Fig.14). A utilização das *flags* permite indicar a existência de fragmentos adicionais, ou seja, se as *flags* tem valor 0 corresponde a que não exista mais fragmentos, enquanto *more fragments* indica que existe mais fragmentos. Por outro lado, o *fragment offset* determina a posição do fragmento, pode ser observar que o MTU corresponde a 1480 bytes para os fragmentos. Portanto, a reconstrução do datagrama original acontecerá quando todos os seus fragmentos forem recebidos (através das *flags*) e, de seguida ordenar os seus fragmentos na sequência correta (através do *fragment offset*) obtendo assim o datagrama original.

No.	Time	Source	Destination	Protoc	Length	BSS Id	Info
1	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=0, ID=e86e) [Reassembled in #4]
2	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=1480, ID=e86e) [Reassembled in #4]
3	0.000000	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=2960, ID=e86e) [Reassembled in #4]
4	0.000000	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=41/10496, ttl=128 (reply in 8)
5	0.003584	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=0, ID=5e5f) [Reassembled in #8]
6	0.003575	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=1480, ID=5e5f) [Reassembled in #8]
7	0.003580	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=2960, ID=5e5f) [Reassembled in #8]
8	0.003584	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=41/10496, ttl=61 (request in 4)
9	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=0, ID=e86f) [Reassembled in #12]
10	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=1480, ID=e86f) [Reassembled in #12]
11	1.009022	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=2960, ID=e86f) [Reassembled in #12]
12	1.009022	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=42/10752, ttl=128 (reply in 16)
13	1.012585	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=0, ID=6192) [Reassembled in #16]
14	1.012604	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=1480, ID=6192) [Reassembled in #16]
15	1.012608	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=2960, ID=6192) [Reassembled in #16]
16	1.012612	193.136.9.240	172.26.42.81	ICMP	823		Echo (ping) reply id=0x0001, seq=42/10752, ttl=61 (request in 12)
17	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=0, ID=e870) [Reassembled in #20]
18	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=1480, ID=e870) [Reassembled in #20]
19	2.022898	172.26.42.81	193.136.9.240	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=2960, ID=e870) [Reassembled in #20]
20	2.022898	172.26.42.81	193.136.9.240	ICMP	823		Echo (ping) request id=0x0001, seq=43/11008, ttl=128 (reply in 24)
21	2.026502	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=0, ID=649d) [Reassembled in #24]
22	2.026521	193.136.9.240	172.26.42.81	IPv4	1514		Fragmented IP protocol (proto=ICMP 1, offset=1480, ID=649d) [Reassembled in #24]

> Frame 4: 823 bytes on wire (6584 bits), 823 bytes captured (6584 bits) on interface \Device\NPF{7078C8DD-4D32-45ED-B5CC-...}	0000 00 00 03 ff 94 00 0c dd 24 31 25 fe 08 00 45 00 \$1%~E
> Ethernet II, Src: IntelCor_31:25:fe (0c:dd:24:31:25:fe), Dst: ComaEnt_ff:94:00 (00:00:03:ff:94:00)	0010 03 29 e8 6e 02 2b 80 01 00 00 ac 1a 2a 51 c1 88 Qj
> Internet Protocol Version 4, Src: 172.26.42.81, Dst: 193.136.9.240	0020 68 69 6a 0b 6c 6d 6e 6f 70 71 72 73 74 75 76 grstuw wabdcfgh
> ... 0101 = Header Length: 20 bytes (5)	0030 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0040 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0050 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0060 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0070 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0080 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0090 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	00a0 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	00b0 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	00c0 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	00d0 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	00e0 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	00f0 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0100 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0110 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0120 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0130 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0140 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0150 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0160 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop
> ... 0101 = Header Length: 20 bytes (5)	0170 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f abcdedfgh ijklnmop

Fig. 14. Reconstrução de uma mensagem ICMP

- f) Sabendo que a opção *-f* (*Windows*) ou *-M* do (*Linux*) ativa a flag “*Don’t Fragment*” (DF) no cabeçalho do IPv4, usando *ping <opção DF> <opção pkt size> SIZE marco.uminho.pt*, (opção *pkt size* = *-l* (*Windows*) ou *-s* (*Linux*)), determine o valor máximo de *SIZE* sem que ocorra fragmentação do pacote? Justifique o valor obtido.

O valor máximo do *SIZE* (tamanho) corresponde a 1472 bytes (Fig.15), sendo que o MTU tem tamanho 1500 bytes. Contudo o cabeçalho do *Internet Protocol* (IP) tem tamanho de 20 bytes. Mais ainda, o ICMP vai também ter um cabeçalho que corresponde a 8 bytes. Logo, serão utilizados em ambos os cabeçalhos, no total 28 bytes, o que corresponde a que o pacote possa ter, no máximo, 1472 bytes sem que ocorra fragmentação (Fig.16).

```
C:\Users\ruben>ping -f -l 1472 marco.uminho.pt

Pinging marco.uminho.pt [193.136.9.240] with 1472 bytes of data:
Reply from 193.136.9.240: bytes=1472 time=2ms TTL=61
Reply from 193.136.9.240: bytes=1472 time=27ms TTL=61
Reply from 193.136.9.240: bytes=1472 time=3ms TTL=61
Reply from 193.136.9.240: bytes=1472 time=2ms TTL=61

Ping statistics for 193.136.9.240:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 2ms, Maximum = 27ms, Average = 8ms
```

Fig. 15. Limite máximo em que o pacote não é fragmentado

```
C:\Users\ruben>ping -f -l 1473 marco.uminho.pt

Pinging marco.uminho.pt [193.136.9.240] with 1473 bytes of data:
Packet needs to be fragmented but DF set.
PING: transmit failed. General failure.
PING: transmit failed. General failure.
PING: transmit failed. General failure.

Ping statistics for 193.136.9.240:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

Fig. 16. Pacote que necessita de fragmentação

(Parte II)

2 Endereçamento e Encaminhamento IP

- 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 mensagem que ilustre de forma clara a topologia definida e o endereçamento usado.

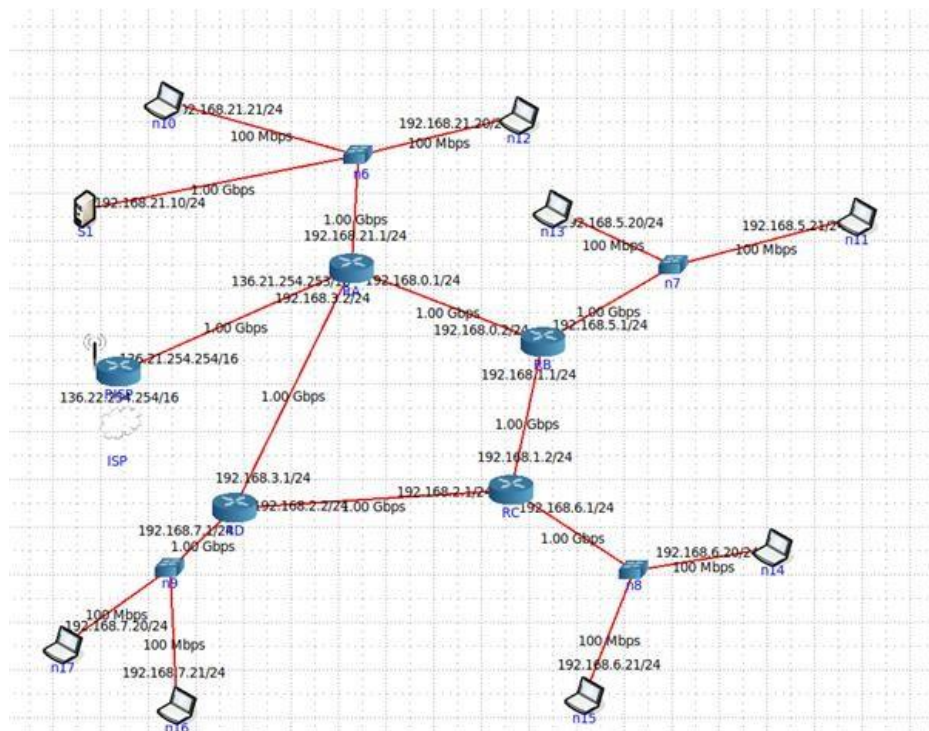


Fig. 17. Topologia CORE

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

Trata-se de endereços privados no qual segundo a norma RFC 1918, os endereços correspondem à classe C denominados por *24-bit blocks*, ou seja, correspondem a máscara de rede 255.255.255.0/24 e de *hosts* 0.0.0.254/24.

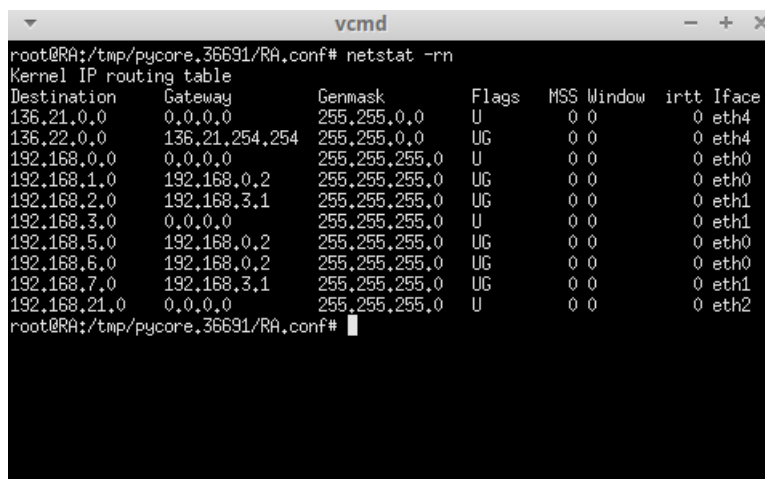
- c) Porque razão o CORE não atribui um endereço IP aos *switches*? Faz sentido na prática um *switch* ter um endereço IP? Justifique.

O CORE não atribui um endereço IP aos *switches*, visto que estes operam na camada inferior ao *Internet Protocol* (IP), ou seja, na segunda camada enquanto o IP opera na terceira camada da pilha protocolar.

2) Para o *router* R_A e o servidor S₁ do departamento A:

- 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*).

As tabelas dos datagramas (Fig.18 e 19) possuem vários parâmetros: na primeira coluna encontra-se o destino (*Destination*); na segunda coluna encontra-se o *Gateway* onde será entregue o datagrama e o redirecionará consoante a rota disponível; na terceira coluna encontramos a máscara de rede onde, neste caso, corresponde a 255.255.255.0, visto que os endereços utilizados são da classe C; na quarta coluna está definida as *Flags* que indicam se a rota se encontra definida (tem como valor UG caso exista uma rota e U quando a rota não está definida); e na última coluna indica a *interface* (*Iface*) utilizada.

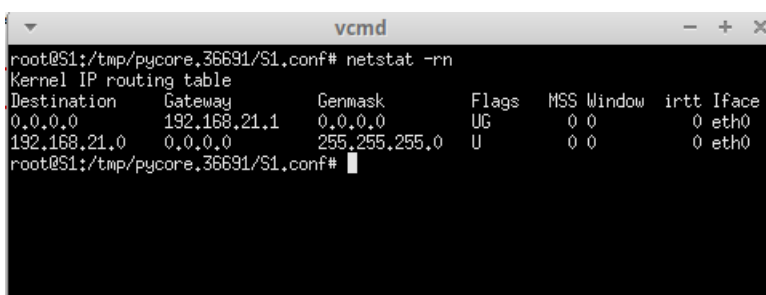


```

root@RA:/tmp/pycore.36691/RA.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt  Iface
136.21.0.0       0.0.0.0         255.255.0.0     U        0  0        0  eth4
136.22.0.0       136.21.254.254  255.255.0.0     UG       0  0        0  eth4
192.168.0.0       0.0.0.0         255.255.255.0   U        0  0        0  eth0
192.168.1.0       192.168.0.2     255.255.255.0   UG       0  0        0  eth0
192.168.2.0       192.168.3.1     255.255.255.0   UG       0  0        0  eth1
192.168.3.0       0.0.0.0         255.255.255.0   U        0  0        0  eth1
192.168.5.0       192.168.0.2     255.255.255.0   UG       0  0        0  eth0
192.168.6.0       192.168.0.2     255.255.255.0   UG       0  0        0  eth0
192.168.7.0       192.168.3.1     255.255.255.0   UG       0  0        0  eth1
192.168.21.0      0.0.0.0         255.255.255.0   U        0  0        0  eth2

```

Fig. 18. Tabela de encaminhamento do *router* R_A



```

root@S1:/tmp/pycore.36691/S1.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt  Iface
0.0.0.0          192.168.21.1    0.0.0.0         UG       0  0        0  eth0
192.168.21.0     0.0.0.0         255.255.255.0   U        0  0        0  eth0

```

Fig. 19. Tabela de encaminhamento do servidor S₁

- b) Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise os processos que estão a correr em cada sistema (*hosts*, *routers*) usando, por exemplo, o comando *ps -ax*).

O encaminhamento é dinâmico, visto que as rotas são definidas através da troca de informação entre os *routers*.

- c) Escolha um PC da organização REDES que esteja o mais distante possível em termos de saltos IP do servidor S₁. Recorrendo apenas ao comando *traceroute -I*, responda às seguintes questões, justificando:
- i) Existe conectividade IP desse PC para o servidor S₁? A quantos saltos IP está o PC selecionado do servidor S₁.

Existe conectividade do PC para o servidor S₁ (Fig.20), executando 4 saltos IP (desprezando o *switch*, visto que este não possui endereço IP).

```

root@n15:/tmp/pycore.36691/n15.conf# traceroute -I 192.168.21.10
traceroute to 192.168.21.10 (192.168.21.10), 30 hops max, 60 byte packets
 1 192.168.6.1 (192.168.6.1) 0.742 ms 0.699 ms 0.696 ms
 2 192.168.1.1 (192.168.1.1) 0.721 ms 0.721 ms 1.148 ms
 3 192.168.0.1 (192.168.0.1) 1.147 ms 1.569 ms 1.780 ms
 4 192.168.21.10 (192.168.21.10) 1.784 ms 1.829 ms 1.829 ms
root@n15:/tmp/pycore.36691/n15.conf#

```

Fig. 20. Comando *traceroute* do PC (n15) para o servidor S₁

- ii) As rotas dos pacotes ICMP *echo reply* são as mesmas, mas em sentido inverso, das rotas dos pacotes ICMP *echo request* trocadas entre esse PC e o servidor S₁? (Sugestão: trace a rota de S₁ para o PC selecionado).

As rotas dos pacotes ICMP *echo request* e *echo reply* são ambas iguais, como se pode observar na figura abaixo (Fig21).

No.	Time	Source	Destination	Protocol	Length	Info
14	15.83284226	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response found)
15	15.83284626	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=2/256, ttl=1 (no response found)
16	15.83284806	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=3/256, ttl=1 (no response found)
17	15.83285197	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=4/256, ttl=2 (no response found)
18	15.83285409	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=5/256, ttl=2 (no response found)
19	15.83285742	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=6/256, ttl=2 (no response found)
20	15.83286033	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=7/256, ttl=3 (no response found)
21	15.83286307	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=8/256, ttl=3 (no response found)
22	15.83286578	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=9/256, ttl=4 (response found)
23	15.83286852	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=10/256, ttl=4 (reply in 41)
24	15.83287125	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=11/256, ttl=4 (reply in 42)
25	15.83287397	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=12/256, ttl=4 (reply in 42)
26	15.83287669	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=13/256, ttl=5 (reply in 44)
27	15.83287942	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=14/256, ttl=5 (reply in 45)
28	15.83288214	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=15/256, ttl=5 (reply in 46)
29	15.83288485	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=16/256, ttl=6 (reply in 50)
30	15.83288758	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
31	15.83289031	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
32	15.83289304	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
33	15.83289577	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
34	15.83289850	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
35	15.83290123	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
36	15.83290396	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
37	15.83290669	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=17/256, ttl=6 (reply in 51)
38	15.83290942	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=18/256, ttl=6 (reply in 52)
39	15.83291215	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
40	15.83291488	192.168.6.21	192.168.21.10	ICMP	102	Time-to-live exceeded Time to live exceeded in transit
41	15.83291761	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=19/256, ttl=6 (reply in 51)
42	15.83292034	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=20/256, ttl=6 (reply in 51)
43	15.83292307	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=21/256, ttl=6 (reply in 51)
44	15.83292580	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=22/256, ttl=6 (reply in 51)
45	15.83292853	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=23/256, ttl=6 (reply in 51)
46	15.83293126	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=24/256, ttl=6 (reply in 51)
47	15.83293399	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=25/256, ttl=6 (reply in 51)
48	15.83293672	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=26/256, ttl=6 (reply in 51)
49	15.83293945	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=27/256, ttl=6 (reply in 51)
50	15.83294218	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=28/256, ttl=6 (reply in 51)
51	15.83294491	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=29/256, ttl=6 (reply in 51)
52	15.83294764	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=30/256, ttl=6 (reply in 51)
53	15.83295037	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=31/256, ttl=6 (reply in 51)
54	15.83295310	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=32/256, ttl=6 (reply in 51)
55	15.83295583	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=33/256, ttl=6 (reply in 51)
56	15.83295856	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=34/256, ttl=6 (reply in 51)
57	15.83296129	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=35/256, ttl=6 (reply in 51)
58	15.83296402	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=36/256, ttl=6 (reply in 51)
59	15.83296675	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=37/256, ttl=6 (reply in 51)
60	15.83296948	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=38/256, ttl=6 (reply in 51)
61	15.83297221	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=39/256, ttl=6 (reply in 51)
62	15.83297494	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=40/256, ttl=6 (reply in 51)
63	15.83297767	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=41/256, ttl=6 (reply in 51)
64	15.83298040	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=42/256, ttl=6 (reply in 51)
65	15.83298313	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=43/256, ttl=6 (reply in 51)
66	15.83298586	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=44/256, ttl=6 (reply in 51)
67	15.83298859	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=45/256, ttl=6 (reply in 51)
68	15.83299132	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=46/256, ttl=6 (reply in 51)
69	15.83299405	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=47/256, ttl=6 (reply in 51)
70	15.83299678	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=48/256, ttl=6 (reply in 51)
71	15.83299951	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=49/256, ttl=6 (reply in 51)
72	15.83300224	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=50/256, ttl=6 (reply in 47)
73	15.83300497	192.168.6.21	192.168.21.10	ICMP	74	Echo (ping) request id=0x001b, seq=51/256, ttl=6 (reply in 48)


```

vcmd
root@S1:/tmp/pycore.36691/S1.conf# route delete 0.0.0.0
SIOCDELRT: No such process
root@S1:/tmp/pycore.36691/S1.conf# route delete default
root@S1:/tmp/pycore.36691/S1.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.21.0   0.0.0.0         255.255.255.0   U        0  0          0 eth0
root@S1:/tmp/pycore.36691/S1.conf#

```

Fig. 22. Comando *route delete* no servidor S₁

```

vcmd
root@S1:/tmp/pycore.36691/S1.conf# ping 192.168.6.20
ping: connect: Network is unreachable
root@S1:/tmp/pycore.36691/S1.conf#

```

Fig. 23. *Ping* sem sucesso para o destino

- e) Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor S₁, por forma a contornar a restrição imposta na alínea anterior. Utilize para o efeito o comando *route add* e registre os comandos que usou.

```

root@S1:/tmp/pycore.36691/S1.conf# route add default gw 192.168.21.1
root@S1:/tmp/pycore.36691/S1.conf# ping 192.168.6.21
PING 192.168.6.21 (192.168.6.21) 56(84) bytes of data:
64 bytes from 192.168.6.21: icmp_seq=1 ttl=61 time=1.17 ms
64 bytes from 192.168.6.21: icmp_seq=2 ttl=61 time=0.921 ms
64 bytes from 192.168.6.21: icmp_seq=3 ttl=61 time=0.857 ms
64 bytes from 192.168.6.21: icmp_seq=4 ttl=61 time=0.201 ms
64 bytes from 192.168.6.21: icmp_seq=5 ttl=61 time=0.532 ms
64 bytes from 192.168.6.21: icmp_seq=6 ttl=61 time=0.278 ms
64 bytes from 192.168.6.21: icmp_seq=7 ttl=61 time=0.872 ms
64 bytes from 192.168.6.21: icmp_seq=8 ttl=61 time=0.864 ms
64 bytes from 192.168.6.21: icmp_seq=9 ttl=61 time=0.873 ms
64 bytes from 192.168.6.21: icmp_seq=10 ttl=61 time=0.917 ms
64 bytes from 192.168.6.21: icmp_seq=11 ttl=61 time=0.239 ms
64 bytes from 192.168.6.21: icmp_seq=12 ttl=61 time=6.05 ms
64 bytes from 192.168.6.21: icmp_seq=13 ttl=61 time=1.23 ms

```

Fig. 24. Comando *route add* no servidor S₁

- f) 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 S₁.

```

root@S1:/tmp/pycore.36691/S1.conf# route add default gw 192.168.21.1
root@S1:/tmp/pycore.36691/S1.conf# ping 192.168.6.21
PING 192.168.6.21 (192.168.6.21) 56(84) bytes of data:
64 bytes from 192.168.6.21: icmp_seq=1 ttl=61 time=1.17 ms
64 bytes from 192.168.6.21: icmp_seq=2 ttl=61 time=0.921 ms
64 bytes from 192.168.6.21: icmp_seq=3 ttl=61 time=0.857 ms
64 bytes from 192.168.6.21: icmp_seq=4 ttl=61 time=0.201 ms
64 bytes from 192.168.6.21: icmp_seq=5 ttl=61 time=0.532 ms
64 bytes from 192.168.6.21: icmp_seq=6 ttl=61 time=0.278 ms
64 bytes from 192.168.6.21: icmp_seq=7 ttl=61 time=0.872 ms
64 bytes from 192.168.6.21: icmp_seq=8 ttl=61 time=0.864 ms
64 bytes from 192.168.6.21: icmp_seq=9 ttl=61 time=0.873 ms
64 bytes from 192.168.6.21: icmp_seq=10 ttl=61 time=0.917 ms
64 bytes from 192.168.6.21: icmp_seq=11 ttl=61 time=0.239 ms
64 bytes from 192.168.6.21: icmp_seq=12 ttl=61 time=6.05 ms
64 bytes from 192.168.6.21: icmp_seq=13 ttl=61 time=1.23 ms

```

Fig. 25. Comando *ping* do servidor S₁ para um PC

- g) Apague nas tabelas de encaminhamento dos *routers* R_A, R_B, R_C e R_D a rede a que pertence a interface *wireless* do router R_{ISP}. Usando rotas por defeito, altere as tabelas de encaminhamento dos equipamentos da organização REDES de forma a possibilitar aos seus utilizadores o acesso genérico à Internet através do router R_{ISP}.
- i) Apresente as novas tabelas de encaminhamento.

```
<A.conf# route del -net 136,21,0,0 netmask 255,255,0,0
root@RA:/tmp/pycore.36691/RA.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
136,22,0,0 136,21,254,254 255,255,0,0 UG 0 0 0 eth4
192,168,0,0 0,0,0,0 255,255,255,0 U 0 0 0 eth0
192,168,1,0 192,168,0,2 255,255,255,0 UG 0 0 0 eth0
192,168,2,0 192,168,3,1 255,255,255,0 UG 0 0 0 eth1
192,168,3,0 0,0,0,0 255,255,255,0 U 0 0 0 eth1
192,168,5,0 192,168,0,2 255,255,255,0 UG 0 0 0 eth0
192,168,6,0 192,168,0,2 255,255,255,0 UG 0 0 0 eth0
192,168,7,0 192,168,3,1 255,255,255,0 UG 0 0 0 eth1
192,168,21,0 0,0,0,0 255,255,255,0 U 0 0 0 eth2
```

Fig. 26. Comando *route delete* no router R_A

```
root@RA:/tmp/pycore.36691/RA.conf# route add default gw 136,21,254,254
root@RA:/tmp/pycore.36691/RA.conf# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 136,21,254,254 0,0,0,0 UG 0 0 0 eth4
136,21,0,0 0,0,0,0 255,255,0,0 U 0 0 0 eth4
136,22,0,0 136,21,254,254 255,255,0,0 UG 2 0 0 eth4
192,168,0,0 0,0,0,0 255,255,255,0 U 0 0 0 eth0
192,168,1,0 192,168,0,2 255,255,255,0 UG 2 0 0 eth0
192,168,2,0 192,168,3,1 255,255,255,0 UG 2 0 0 eth1
192,168,3,0 0,0,0,0 255,255,255,0 U 0 0 0 eth1
192,168,5,0 192,168,0,2 255,255,255,0 UG 2 0 0 eth0
192,168,6,0 192,168,0,2 255,255,255,0 UG 3 0 0 eth0
192,168,7,0 192,168,3,1 255,255,255,0 UG 2 0 0 eth1
192,168,21,0 0,0,0,0 255,255,255,0 U 0 0 0 eth2
root@RA:/tmp/pycore.36691/RA.conf#
```

Fig. 27. Comando *route add* no router R_A

```
vcmd
<B.conf# route del -net 136,21,0,0 netmask 255,255,0,0
root@RB:/tmp/pycore.36691/RB.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
136,22,0,0 192,168,0,1 255,255,0,0 UG 0 0 0 eth0
192,168,0,0 0,0,0,0 255,255,255,0 U 0 0 0 eth0
192,168,1,0 0,0,0,0 255,255,255,0 U 0 0 0 eth1
192,168,2,0 192,168,1,2 255,255,255,0 UG 0 0 0 eth1
192,168,3,0 192,168,0,1 255,255,255,0 UG 0 0 0 eth0
192,168,5,0 0,0,0,0 255,255,255,0 U 0 0 0 eth2
192,168,6,0 192,168,1,2 255,255,255,0 UG 0 0 0 eth1
192,168,7,0 192,168,0,1 255,255,255,0 UG 0 0 0 eth0
192,168,21,0 192,168,0,1 255,255,255,0 UG 0 0 0 eth0
root@RB:/tmp/pycore.36691/RB.conf#
```

Fig. 28. Comando *route delete* no router R_B

```
root@RB:/tmp/pycore.36691/RB.conf# route add default gw 192,168,0,1
root@RB:/tmp/pycore.36691/RB.conf# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 192,168,0,1 0,0,0,0 UG 0 0 0 eth0
136,21,0,0 192,168,0,1 255,255,0,0 UG 2 0 0 eth0
136,22,0,0 192,168,0,1 255,255,0,0 UG 3 0 0 eth0
192,168,0,0 0,0,0,0 255,255,255,0 U 0 0 0 eth0
192,168,1,0 0,0,0,0 255,255,255,0 U 0 0 0 eth1
192,168,2,0 192,168,1,2 255,255,255,0 UG 2 0 0 eth1
192,168,3,0 192,168,0,1 255,255,255,0 UG 2 0 0 eth0
192,168,5,0 0,0,0,0 255,255,255,0 U 0 0 0 eth2
192,168,6,0 192,168,1,2 255,255,255,0 UG 2 0 0 eth1
192,168,7,0 192,168,0,1 255,255,255,0 UG 3 0 0 eth0
192,168,21,0 192,168,0,1 255,255,255,0 UG 2 0 0 eth0
root@RB:/tmp/pycore.36691/RB.conf#
```

Fig. 29. Comando *route add* no router R_B

```

vcmd
<conf# route del -net 136.21.0.0 netmask 255.255.0.0
root@RC:/tmp/pycore.36691/RC.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
136.22.0.0 192.168.1.1 255.255.0.0 UG 0 0 0 eth0
192.168.0.0 192.168.1.1 255.255.255.0 UG 0 0 0 eth0
192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
192.168.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
192.168.3.0 192.168.2.2 255.255.255.0 UG 0 0 0 eth1
192.168.5.0 192.168.1.1 255.255.255.0 UG 0 0 0 eth0
192.168.6.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
192.168.7.0 192.168.2.2 255.255.255.0 UG 0 0 0 eth1
192.168.21.0 192.168.1.1 255.255.255.0 UG 0 0 0 eth0
root@RC:/tmp/pycore.36691/RC.conf#

```

Fig. 30. Comando *route delete* no router Rc

```

root@RC:/tmp/pycore.36691/RC.conf# route add default gw 192.168.2.2
root@RC:/tmp/pycore.36691/RC.conf# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 192.168.2.2 0.0.0.0 UG 0 0 0 eth1
136.21.0.0 192.168.1.1 255.255.0.0 UG 3 0 0 eth0
136.22.0.0 192.168.1.1 255.255.0.0 UG 4 0 0 eth0
192.168.0.0 192.168.1.1 255.255.255.0 UG 2 0 0 eth0
192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
192.168.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
192.168.3.0 192.168.2.2 255.255.255.0 UG 2 0 0 eth1
192.168.5.0 192.168.1.1 255.255.255.0 UG 2 0 0 eth0
192.168.6.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
192.168.7.0 192.168.2.2 255.255.255.0 UG 2 0 0 eth1
192.168.21.0 192.168.1.1 255.255.255.0 UG 3 0 0 eth0
root@RC:/tmp/pycore.36691/RC.conf#

```

Fig. 31. Comando *route add* no router Rc

```

vcmd
<conf# route del -net 136.21.0.0 netmask 255.255.0.0
root@RD:/tmp/pycore.36691/RD.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
136.22.0.0 192.168.3.2 255.255.0.0 UG 0 0 0 eth1
192.168.0.0 192.168.3.2 255.255.255.0 UG 0 0 0 eth1
192.168.1.0 192.168.2.1 255.255.255.0 UG 0 0 0 eth0
192.168.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
192.168.3.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
192.168.5.0 192.168.2.1 255.255.255.0 UG 0 0 0 eth0
192.168.6.0 192.168.2.1 255.255.255.0 UG 0 0 0 eth0
192.168.7.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
192.168.21.0 192.168.3.2 255.255.255.0 UG 0 0 0 eth1
root@RD:/tmp/pycore.36691/RD.conf#

```

Fig. 32. Comando *route delete* no router Rd

```

root@RD:/tmp/pycore.36691/RD.conf# route add default gw 192.168.3.2
root@RD:/tmp/pycore.36691/RD.conf# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
default 192.168.3.2 0.0.0.0 UG 0 0 0 eth1
136.21.0.0 192.168.3.2 255.255.0.0 UG 2 0 0 eth1
136.22.0.0 192.168.3.2 255.255.0.0 UG 3 0 0 eth1
192.168.0.0 192.168.3.2 255.255.255.0 UG 2 0 0 eth1
192.168.1.0 192.168.2.1 255.255.255.0 UG 2 0 0 eth0
192.168.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
192.168.3.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
192.168.5.0 192.168.2.1 255.255.255.0 UG 3 0 0 eth0
192.168.6.0 192.168.2.1 255.255.255.0 UG 2 0 0 eth0
192.168.7.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
192.168.21.0 192.168.3.2 255.255.255.0 UG 2 0 0 eth1
root@RD:/tmp/pycore.36691/RD.conf#

```

Fig. 33. Comando *route add* no router Rd

ii) Teste a solução verificando a conectividade entre PC de cada departamento e a interface *wireless* do router R_{ISP} (basta usar um PC por departamento).

```

root@n10:/tmp/pycore.36691/n10.conf# ping 136.21.254.254
PING 136.21.254.254 (136.21.254.254) 56(84) bytes of data.
64 bytes from 136.21.254.254: icmp_seq=1 ttl=63 time=0.645 ms
64 bytes from 136.21.254.254: icmp_seq=2 ttl=63 time=0.147 ms
64 bytes from 136.21.254.254: icmp_seq=3 ttl=63 time=0.467 ms

```

Fig. 34. Comando *ping* de um PC do departamento A para o router R_{ISP}

```

root@n11:/tmp/pycore.36691/n11.conf# ping 136.21.254.254
PING 136.21.254.254 (136.21.254.254) 56(84) bytes of data.
64 bytes from 136.21.254.254: icmp_seq=1 ttl=62 time=0.789 ms
64 bytes from 136.21.254.254: icmp_seq=2 ttl=62 time=0.180 ms
64 bytes from 136.21.254.254: icmp_seq=3 ttl=62 time=0.731 ms

```

Fig. 35. Comando *ping* de um PC do departamento B para o router R_{ISP}

```

root@n15:/tmp/pycore.36691/n15.conf# ping 136.21.254.254
PING 136.21.254.254 (136.21.254.254) 56(84) bytes of data.
64 bytes from 136.21.254.254: icmp_seq=1 ttl=61 time=1.09 ms
64 bytes from 136.21.254.254: icmp_seq=2 ttl=61 time=0.928 ms
64 bytes from 136.21.254.254: icmp_seq=3 ttl=61 time=0.883 ms
64 bytes from 136.21.254.254: icmp_seq=4 ttl=61 time=0.260 ms
64 bytes from 136.21.254.254: icmp_seq=5 ttl=61 time=0.196 ms

```

Fig. 36. Comando *ping* de um PC do departamento C para o router R_{ISP}

```

root@n16:/tmp/pycore.36691/n16.conf# ping 136.21.254.254
PING 136.21.254.254 (136.21.254.254) 56(84) bytes of data.
64 bytes from 136.21.254.254: icmp_seq=1 ttl=62 time=0.600 ms
64 bytes from 136.21.254.254: icmp_seq=2 ttl=62 time=0.184 ms
64 bytes from 136.21.254.254: icmp_seq=3 ttl=62 time=0.649 ms
64 bytes from 136.21.254.254: icmp_seq=4 ttl=62 time=0.890 ms

```

Fig. 37. Comando *ping* de um PC do departamento D para o router R_{ISP}

3 Definição de Sub-redes

- 1) Considere que dispõe apenas do endereço da rede IP 150.21.80.0/20, em que 21 é o decimal correspondendo ao número do grupo (G21). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo as redes de acesso e central (*core*) inalteradas) e atribua endereços às interfaces dos vários sistemas envolvidos. Assuma que todos os endereços de sub-redes são usáveis. Deve justificar as opções usadas.

Tendo em conta os quatro departamentos existentes, são necessários, no mínimo 4 (2^2) subredes, uma para cada departamento. Porém, de modo a garantir a possibilidade de expansão da organização com novos departamentos, decidiu-se utilizar mais um bit, ou seja, 3 bits para *subnetting* permitindo um total de 8 (2^3) subredes (Fig.38).

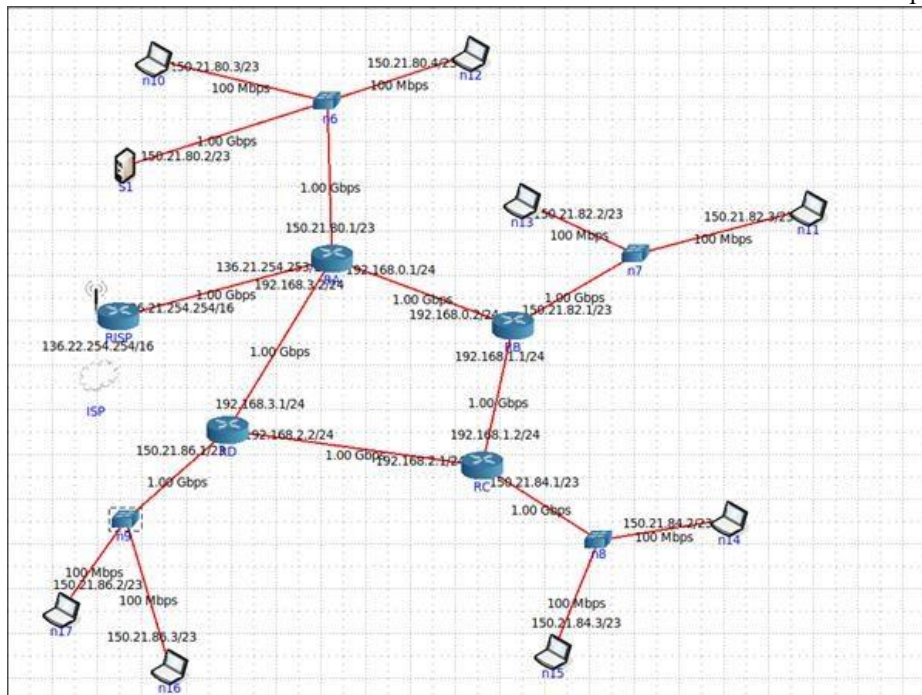


Fig. 38. Topologia da organização REDES

- 2) Qual é a máscara de rede que usou (em formato decimal)? Quantos *hosts* IP pode interligar no máximo em cada departamento? Quantos prefixos de sub-rede ficam disponíveis para uso futuro? Justifique.

A máscara de rede utilizada foi /23 (255.255.254.0), em que no formato binário corresponde a (11111111.11111111.11111110.00000000).

Em cada subrede tem como possibilidade 510 *hosts* diferentes, calculado através $2^9 - 2 = 510$ (2 representa os dois *hosts* reservados). Na eventual expansão de novos departamentos encontra-se disponíveis 4 subredes: 150.21.88.0/23, 150.21.90.0/23, 150.21.92.0/23, 150.21.94.0/23.

- 3) Garanta e verifique que a conectividade IP entre as várias redes locais da organização REDES é mantida. Explique como procedeu.

Para verificar a conectividade IP entre as várias redes locais, utilizou-se o comando *ping* entre os diversos departamentos. Através das figuras abaixo, pode-se confirmar que a conectividade é mantida na organização REDES.

```
vcmd
root@n10:/tmp/pycore.46177/n10.conf# 150.21.84.2
150.21.84.2: command not found
root@n10:/tmp/pycore.46177/n10.conf# ping 150.21.84.2
PING 150.21.84.2 (150.21.84.2) 56(84) bytes of data:
64 bytes from 150.21.84.2: icmp_seq=1 ttl=61 time=1.13 ms
64 bytes from 150.21.84.2: icmp_seq=2 ttl=61 time=0.989 ms
64 bytes from 150.21.84.2: icmp_seq=3 ttl=61 time=0.273 ms
64 bytes from 150.21.84.2: icmp_seq=4 ttl=61 time=0.236 ms
64 bytes from 150.21.84.2: icmp_seq=5 ttl=61 time=0.202 ms
64 bytes from 150.21.84.2: icmp_seq=6 ttl=61 time=0.239 ms
64 bytes from 150.21.84.2: icmp_seq=7 ttl=61 time=0.842 ms
```

Fig. 39. Conetividade do departamento A

```

vcmd
root@n13:/tmp/pycore.46177/n13.conf# ping 150.21.86,2
PING 150.21.86,2 (150.21.86,2) 56(84) bytes of data.
64 bytes from 150.21.86,2: icmp_seq=1 ttl=61 time=0,927 ms
64 bytes from 150.21.86,2: icmp_seq=2 ttl=61 time=0,429 ms
64 bytes from 150.21.86,2: icmp_seq=3 ttl=61 time=0,913 ms
64 bytes from 150.21.86,2: icmp_seq=4 ttl=61 time=0,981 ms
64 bytes from 150.21.86,2: icmp_seq=5 ttl=61 time=0,217 ms
64 bytes from 150.21.86,2: icmp_seq=6 ttl=61 time=1,26 ms
64 bytes from 150.21.86,2: icmp_seq=7 ttl=61 time=0,369 ms
64 bytes from 150.21.86,2: icmp_seq=8 ttl=61 time=0,971 ms
64 bytes from 150.21.86,2: icmp_seq=9 ttl=61 time=0,448 ms
64 bytes from 150.21.86,2: icmp_seq=10 ttl=61 time=0,225 ms
64 bytes from 150.21.86,2: icmp_seq=11 ttl=61 time=0,789 ms
64 bytes from 150.21.86,2: icmp_seq=12 ttl=61 time=0,230 ms
64 bytes from 150.21.86,2: icmp_seq=13 ttl=61 time=0,929 ms

```

Fig. 40. Conetividade do departamento B

```

vcmd
root@n15:/tmp/pycore.46177/n15.conf# ping 150.21.86,3
PING 150.21.86,3 (150.21.86,3) 56(84) bytes of data.
64 bytes from 150.21.86,3: icmp_seq=1 ttl=62 time=1,15 ms
64 bytes from 150.21.86,3: icmp_seq=2 ttl=62 time=0,210 ms
64 bytes from 150.21.86,3: icmp_seq=3 ttl=62 time=0,181 ms

```

Fig. 41. Conetividade do departamento C

```

vcmd
root@n17:/tmp/pycore.46177/n17.conf# ping 150.21.82,3
PING 150.21.82,3 (150.21.82,3) 56(84) bytes of data.
64 bytes from 150.21.82,3: icmp_seq=1 ttl=61 time=0,824 ms
64 bytes from 150.21.82,3: icmp_seq=2 ttl=61 time=0,257 ms
64 bytes from 150.21.82,3: icmp_seq=3 ttl=61 time=0,231 ms

```

Fig. 42. Conetividade do departamento D

4 Conclusão

As diferentes vertentes da *Internet Protocol* (IP) abordadas neste trabalho prático tiveram como base os conhecimentos teóricos adquiridos pelos autores nas aulas lecionadas da UC (Unidade Curricular). Deste modo, foram exploradas as características da IP, nomeadamente a fragmentação de datagramas, endereçamento e encaminhamento IP com o auxílio do CORE e do *Wireshark*. Contudo, o trabalho apresentou alguns desafios, que no final foram superados. Em suma, esta temática proporcionou grande interesse e experiência prática aos autores, visando mostrar ao docente os conhecimentos adquiridos nas aulas práticas, elevando no processo o grau de conhecimento dos autores.