

Trabalho Prático Nº2 - Protocolo IPv4

Simão Cunha^[a93262], Duarte Leitão^[a100550], and Diogo Barros^[a100600]

Universidade do Minho - Campus de Gualtar, R. da Universidade, 4710-057 Braga Portugal

Redes de Computadores (2022/2023) - PL10 - Grupo 7

Parte I: Datagramas IP e Fragmentação

1

- 1.1 Active o Wireshark no host Lost. Numa shell de Lost execute o comando `traceroute -I` para o endereço IP do Found. Registe e analise o tráfego ICMP enviado pelo sistema Lost e o tráfego ICMP recebido como resposta. Explique os resultados obtidos tendo em conta o princípio de funcionamento do traceroute

Para resolvermos este exercício, o primeiro passo consistiu em ligar o Wireshark no host Lost e executar o comando `traceroute` numa shell deste mesmo nó. A figura 1 mostra esse processo, mostrando os diversos saltos (*hops*) executados de um *end system* até ao outro.

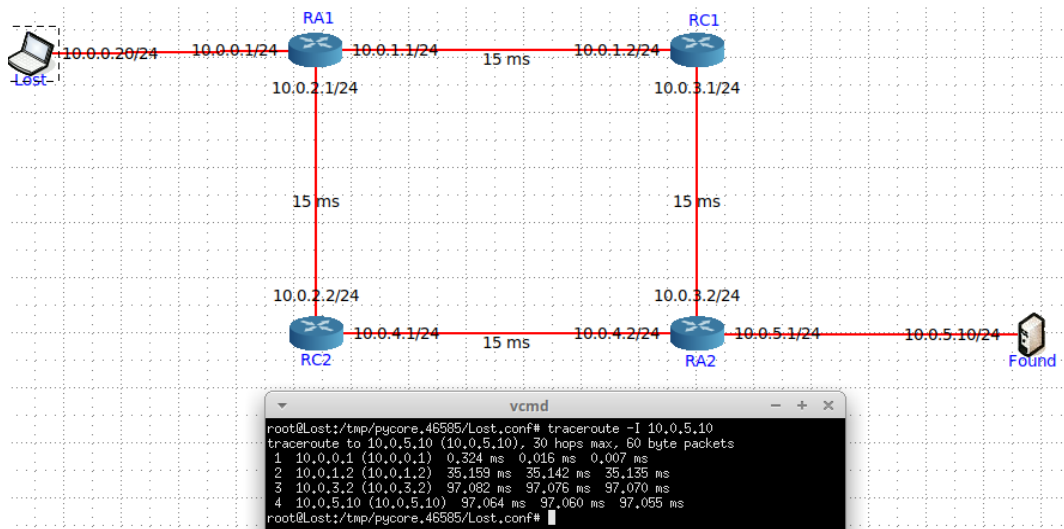


Figura 1: Comando `traceroute -I 10.0.5.10` executado

Durante a execução do comando acima, analisamos o tráfego ICMP recebido como resposta com o Wireshark, tal como se observa abaixo:

No.	Time	Source	Destination	Protocol	Length	Info
10	12.008234118	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
11	12.914862721	00:00:00:aa:00:08	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.20
12	12.914867797	00:00:00:aa:00:09	00:00:00:aa:00:08	ARP	42	10.0.0.1 is at 00:00:00:aa:00:09
13	12.914902131	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response found!)
14	12.915024083	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
15	12.915033737	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=2/512, ttl=1 (no response found!)
16	12.915040755	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response found!)
17	12.915048114	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no response found!)
18	12.915055941	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no response found!)
19	12.915058268	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no response found!)
20	12.915061766	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no response found!)
21	12.915064351	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no response found!)
22	12.915067322	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no response found!)
23	12.915070902	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in 4s)
24	12.915073552	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in 4s)
25	12.915076670	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (reply in 4s)
26	12.915081593	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (reply in 4s)
27	12.915084397	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (reply in 4s)
28	12.915088964	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (reply in 4s)
29	12.915090957	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (reply in 4s)
30	12.915093265	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=17/4352, ttl=6 (reply in 4s)
31	12.915096347	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=18/4608, ttl=6 (reply in 4s)
32	12.915098924	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=19/4864, ttl=7 (reply in 4s)
33	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
34	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
35	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
36	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
37	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
38	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
39	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
40	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
41	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
42	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
43	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
44	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
45	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
46	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
47	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
48	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
49	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
50	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
51	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
52	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
53	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
54	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
55	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
56	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
57	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
58	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
59	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
60	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
61	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
62	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
63	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
64	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
65	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
66	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
67	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
68	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
69	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
70	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
71	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
72	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
73	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
74	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
75	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
76	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
77	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
78	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
79	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
80	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
81	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
82	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
83	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
84	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
85	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
86	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
87	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
88	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
89	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
90	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
91	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
92	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
93	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
94	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
95	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
96	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
97	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
98	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
99	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
100	12.915098450	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)

Figura 2: Tráfego registado com o Wireshark

Da imagem 2, podemos retirar algumas conclusões:

- a origem é o host Lost (10.0.0.20) e o destino dos pacotes é o host Found (10.0.5.10);
- o TTL (time to live) vai incrementando a cada 3 pacotes enviados;
- a partir de TTL = 4, o host Lost começa a receber respostas do Found - tal como se comprova pelo pacote nº 25 capturado.

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

Uma vez que o host Lost começa a receber respostas do Found com TTL=4, o valor inicial mínimo para este campo TTL será 4. De forma a comprovar este facto, consultamos o campo *Time to live* presente na informação da pilha protocolar do pacote capturado através do Wireshark, e lá está realmente o valor 4.

No.	Time	Source	Destination	Protocol	Length	Info
10	12.008234118	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
11	12.914862721	00:00:00:aa:00:08	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.20
12	12.914867797	00:00:00:aa:00:09	00:00:00:aa:00:08	ARP	42	10.0.0.1 is at 00:00:00:aa:00:09
13	12.914902131	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response found!)
14	12.915024083	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
15	12.915033737	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=2/512, ttl=1 (no response found!)
16	12.915040755	10.0.0.20	10.0.5.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
17	12.915048114	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
18	12.915055941	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response found!)
19	12.915058268	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no response found!)
20	12.915061766	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no response found!)
21	12.915064351	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no response found!)
22	12.915067322	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no response found!)
23	12.915070902	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no response found!)
24	12.915073552	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no response found!)
25	12.915076670	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in 4s)
26	12.915079552	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in 4s)
▶ Frame 25: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface veth1.0.4c, id 0 ▶ Ethernet II, Src: 00:00:00:aa:00:08 (00:00:00:aa:00:08), Dst: 00:00:00:aa:00:09 (00:00:00:aa:00:09) ▶ Internet Protocol Version 4, Src: 10.0.0.20, Dst: 10.0.5.10 0100 = Version: 4 0101 = Header Length: 20 bytes (5) ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 68 Identification: 0x9dd7 (40407) ▶ Flags: 0x0000 Fragment offset: 0 ▶ Time to live: 4 Protocol: ICMP (1) Header checksum: 0xffff (validation disabled) [Header checksum status: Unverified] Source: 10.0.0.20 Destination: 10.0.5.10 ▶ Internet Control Message Protocol						

Figura 3: Pacote capturado

1.3 Calcule o valor médio do tempo de ida-e-volta (RTT - Round-Trip Time) obtido no acesso ao servidor. Por modo a obter uma média mais confiável, poderá alterar o número pacotes de prova com a opção -q.

De modo a obtemos uma média mais confiável no cálculo do RTT no acesso ao servidor, alteramos o número de pacotes de prova para o máximo possível (i.e. 10 pacotes). A figura 4 mostra a obtenção dos valores obtidos pelo traceroute.

```
root@lost:/tmp/pycore.46585/lost.conf# traceroute -I 10.0.5.10 -q 12
no more than 10 probes per hop
root@lost:/tmp/pycore.46585/lost.conf# traceroute -I 10.0.5.10 -q 10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0,050 ms 0,005 ms 0,005 ms 0,005 ms 0,005 ms 0,004 ms * * * *
 2 10.0.1.2 (10.0.1.2) 32,440 ms 32,425 ms 32,422 ms 32,418 ms 32,413 ms 32,407 ms * * * *
 3 10.0.3.2 (10.0.3.2) 64,266 ms 64,263 ms 64,719 ms 64,689 ms 64,685 ms 64,681 ms * * * *
 4 10.0.5.10 (10.0.5.10) 69,389 ms 69,388 ms 67,133 ms 67,118 ms 63,217 ms 63,199 ms 63,197 ms 63,194 ms 73,173 ms 73,156 ms
root@lost:/tmp/pycore.46585/lost.conf#
```

Figura 4: Alteração dos pacotes de prova com o traceroute

Assim, podemos finalmente calcular o RTT:

$$\frac{69.389 + 69.388 + 67.133 + 67.118 + 63.217 + 63.199 + 63.197 + 63.194 + 73.173 + 73.156}{10} = 67.216 \text{ ms}$$

Depois de algumas execuções do comando traceroute, reparamos que os valores no último salto variavam de forma significativa entre diferentes execuções. Acreditamos que isto possa ser explicado pelo emulador Core a demonstrar o congestionamento que existe na rede.

1.4 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 numa rede real?

Não será possível obter o valor médio do atraso num só sentido (One-Way Delay). Isto deve-se pelo facto do número de nodos poder ser diferente entre a rota de saída e de entrada, o que implica a existência de percursos diferentes. Além disso, existe o congestionamento na rede, o que nos leva a concluir que um caminho teoricamente mais rápido (com menos *hops*) poderá ser mais lento que outro com mais saltos. No emulador Core, poderá ser viável calcular o RTT através do quociente entre este valor e o valor 2, mas no mundo real não será.

2

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

180	23.271061364	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request id=0x0001, seq=15/3840, ttl=5 (reply i
181	23.271077441	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request id=0x0001, seq=16/4096, ttl=6 (reply i
182	23.273276293	172.16.2.1	172.26.12.233	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
183	23.273276594	172.26.254.254	172.26.12.233	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
184	23.273276701	172.16.2.1	172.26.12.233	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
185	23.273276807	172.26.254.254	172.26.12.233	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
186	23.273276910	172.16.2.1	172.26.12.233	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
187	23.273277015	172.26.254.254	172.26.12.233	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
188	23.273277125	193.136.9.240	172.26.12.233	ICMP	74 Echo (ping) reply id=0x0001, seq=10/2560, ttl=61 (reques
189	23.273277231	172.16.115.252	172.26.12.233	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
190	23.273360582	193.136.9.240	172.26.12.233	ICMP	74 Echo (ping) reply id=0x0001, seq=15/3840, ttl=61 (reques
191	23.273360694	193.136.9.240	172.26.12.233	ICMP	74 Echo (ping) reply id=0x0001, seq=16/4096, ttl=61 (reques

Figura 5: Tráfego criado através do comando traceroute para marco.uminho.pt

Nesta alínea, enviamos um comando traceroute da nossa máquina nativa para `marco.uminho.pt` e capturamos o tráfego ICMP através do Wireshark. Ora, na figura 5, podemos observar que existem *echo requests* e *echo reply's*. No primeiro caso, a origem será o endereço IP da interface ativa da nossa máquina e o destino será o endereço IP de `marco.uminho.pt` - no 2º caso a ordem inverte por se tratar de uma resposta.

Assim o endereço IP da interface ativa do nosso computador é 172.26.12.233.

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

No.	Time	Source	Destination	Protocol	Length	Info
166	23.270827156	172.26.12.233	193.136.9.240	ICMP	74	Echo (ping) request id=0x0001, seq=0x0001
167	23.270875209	172.26.12.233	193.136.9.240	ICMP	74	Echo (ping) request id=0x0001, seq=0x0001
168	23.270892298	172.26.12.233	193.136.9.240	ICMP	74	Echo (ping) request id=0x0001, seq=0x0001
169	23.270910658	172.26.12.233	193.136.9.240	ICMP	74	Echo (ping) request id=0x0001, seq=0x0001

▶ Frame 180: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlp2s0, id 0 ▶ Ethernet II, Src: IntelCor_a5:1e:dc (38:ba:f8:a5:1e:dc), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00) ▶ Internet Protocol Version 4, Src: 172.26.12.233, Dst: 193.136.9.240 0100 = Version: 4 0101 = Header Length: 20 bytes (5) ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x9e14 (40468) ▶ Flags: 0x00 ...0 0000 0000 0000 = Fragment Offset: 0 Time to Live: 5 Protocol: ICMP (1) Header Checksum: 0x9331 [validation disabled] [Header checksum status: Unverified] Source Address: 172.26.12.233 Destination Address: 193.136.9.240 ▶ Internet Control Message Protocol

Figura 6: Visualização do campo *protocol* em pacote ICMP

Tal como observado na figura 6, o valor observado no campo *protocol* é ICMP, protocolo usado para enviar a mensagem de informação ao nível de problemas na comunicação na rede (*error reporting*) da origem até ao destino.

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

```

0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0x9e14 (40468)
▶ Flags: 0x00
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 5

```

Figura 7: Visualização dos tamanhos do cabeçalho IPv4 e do pacote em concreto

Podemos observar na figura 7 dois valores importantes: o tamanho do pacote (*total length*, que contém o valor 60) e o tamanho do cabeçalho IPv4 (*Header length*, que contém o valor 20). Assim, de forma a calcular o valor do *payload*, basta subtrair estes dois valores. Ou seja,

$$\text{payload} = (\text{tamanho total da mensagem}) - (\text{header IPv4}) = 60 - 20 = 40 \text{ bytes.}$$

2.4 O datagrama IP foi fragmentado? Justifique.

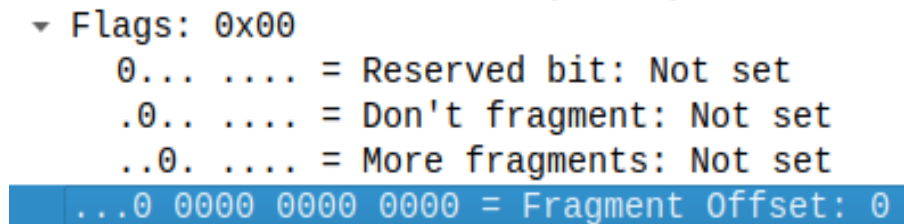


Figura 8: Visualização das *flags* da fragmentação

Na figura 8, podemos ver que o campo *Fragment offset* tem o valor 0, o que nos permite concluir que ou este datagrama não foi fragmentado ou, então, este é o primeiro fragmento.

Além disso, também observamos que a *flag More fragments* tem valor 0 (Not set), o que, juntamente com o campo *Fragment offset*, nos permitem concluir que, não havendo mais fragmentos, o datagrama IP não foi fragmentado.

2.5 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.

De acordo com a nossa captura no Wireshark verificada em 10, verificamos que os campos *identification*, o *header checksum* e o *time to leave (TTL)* do cabeçalho IP variam de pacote para pacote.

166	23.270827156	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=1/256, ttl=1 (no respon
167	23.270875209	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=2/512, ttl=1 (no respon
168	23.270892298	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=3/768, ttl=1 (no respon
169	23.270910658	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=4/1024, ttl=2 (no respo
170	23.270924875	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=5/1280, ttl=2 (no respo
171	23.270937663	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=6/1536, ttl=2 (no respo
172	23.270955583	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=7/1792, ttl=3 (no respo
173	23.270968061	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=8/2048, ttl=3 (no respo
174	23.270980274	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=9/2304, ttl=3 (no respo
175	23.270995672	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=10/2560, ttl=4 (reply i
176	23.271008904	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=11/2816, ttl=4 (no resp
177	23.271021157	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=12/3072, ttl=4 (no resp
178	23.271036655	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=13/3328, ttl=5 (no resp
179	23.271049873	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=14/3584, ttl=5 (no resp
180	23.271061364	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=15/3840, ttl=5 (reply i
181	23.271077441	172.26.12.233	193.136.9.240	ICMP	74 Echo (ping) request	id=0x0001, seq=16/4096, ttl=6 (reply i

Figura 9: Ordenação dos pacotes por *Source*

No.	Time	Source	Destination	Protocol	Length	Info
166	23.270827156	172.26.12.233	193.136.9.240	ICMP	74	Echo (ping) request id=0x0001, seq=1/256, ttl=1 (no respon
193	23.273565924	172.16.115.252	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
192	23.273379558	172.16.115.252	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
191	23.273360694	193.136.9.240	172.26.12.233	ICMP	74	Echo (ping) reply id=0x0001, seq=16/4896, ttl=61 (reques
190	23.273360582	193.136.9.240	172.26.12.233	ICMP	74	Echo (ping) reply id=0x0001, seq=15/3840, ttl=61 (reques
189	23.273277231	172.16.115.252	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
188	23.273277125	193.136.9.240	172.26.12.233	ICMP	74	Echo (ping) reply id=0x0001, seq=10/2560, ttl=61 (reques
187	23.273277015	172.26.254.254	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
186	23.273276910	172.16.2.1	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
185	23.273276807	172.26.254.254	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
184	23.273276701	172.16.2.1	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
183	23.273276594	172.26.254.254	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
182	23.273276293	172.16.2.1	172.26.12.233	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)

Frame 186: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface wlp2s0, id 0						
Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: IntelCor_a5:1e:dc (38:ba:f8:a5:1e:dc)						
Internet Protocol Version 4, Src: 172.16.2.1, Dst: 172.26.12.233						
0100 = Version: 4						
... 0101 = Header Length: 20 bytes (5)						
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
Total Length: 56						
Identification: 0x8934 (35124)						
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: 254						

Frame 185: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface wlp2s0, id 0						
Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: IntelCor_a5:1e:dc (38:ba:f8:a5:1e:dc)						
Internet Protocol Version 4, Src: 172.26.254.254, Dst: 172.26.12.233						
0100 = Version: 4						
... 0101 = Header Length: 20 bytes (5)						
Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)						
Total Length: 56						
Identification: 0x3d7f (15871)						
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: 255						

Figura 11: Tráfego capturado por destino

(a) Qual é o valor do campo TTL recebido no seu computador? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL Exceeded recebidas no seu computador? Porquê? Tal como observamos em 11, o valor do campo TTL exceeded varia entre 253 e 255. Este valor não permanece constante para todas as mensagens de resposta ICMP TTL Exceeded, porque pacotes diferentes podem (ou não) tomar rotas diferentes, levando a um decréscimo mais ou menos acentuado que outros pacotes.

(b) Porque razão as mensagens de resposta ICMP TTL Exceeded são sempre enviadas na origem com um valor relativamente alto? As mensagens de resposta ICMP TTL Exceeded são sempre enviadas na origem com um valor relativamente alto para evitar que os pacotes sejam descartados demasiado cedo por um router por causa de um valor muito baixo do TTL, levando a que descarte o mesmo ainda antes de chegar ao seu destino final. Caso o valor da mensagem de resposta ICMP TTL Exceeded, seja mais baixa que o valor do TTL, esta mensagem é enviada de volta ao host de origem, e o campo TTL do cabeçalho IP é atualizado para refletir o número de saltos necessários para alcançar o dispositivo que descartou o pacote. Isso é feito para que o host de origem possa determinar com mais precisão o número de saltos necessários para alcançar o destino pretendido.

2.8 Sabendo que o ICMP é um protocolo pertencente ao nível de rede, discuta se a informação contida no cabeçalho ICMP poderia ser incluída no cabeçalho IPv4? Quais seriam as vantagens/desvantagens resultantes dessa hipotética inclusão?

Embora o protocolo ICMP seja um protocolo pertencente ao nível de rede, este protocolo é usado para enviar mensagens de controlo de erros e de estado entre dispositivos de rede e o protocolo IPv4 é usado para transportar os pacotes de dados pela rede.

Uma vantagem provável vinda da inclusão de informações do cabeçalho ICMP no cabeçalho IPv4 seria a de maior eficiência e simplicidade na implementação de protocolos de rede. No entanto, essa vantagem não é significativa o suficiente para justificar a inclusão de informações do cabeçalho ICMP no cabeçalho IPv4, pois iria aumentar um overhead protocolar demasiado pesado.

A principal desvantagem é que a inclusão dessas informações no cabeçalho IPv4 aumentaria o tamanho do pacote, o que pode levar a fragmentação do pacote em redes com um MTU menor. Além disso, a inclusão dessas informações no cabeçalho IPv4 tornaria o processamento do pacote mais complexo e aumentaria o overhead protocolar.

3

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

```
user@vivobook [~] ○ [17:17:30]
> traceroute -I marco.uminho.pt 3607
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 3607 byte packets
 1 _gateway (172.26.254.254)  2.968 ms  18.779 ms  18.737 ms
 2 172.16.2.1 (172.16.2.1)  18.695 ms  18.658 ms  18.618 ms
 3 172.16.115.252 (172.16.115.252)  18.576 ms  18.538 ms  18.497 ms
 4 marco.uminho.pt (193.136.9.240)  18.557 ms  19.269 ms  19.229 ms
```

Figura 12: Traceroute com a definição do número de bytes enviados no campo de dados do pacote ICMP

No.	Time	Source	Destination	Protocol	Length	Info
5	0.005729937	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no respon
8	0.005799295	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=2/512, ttl=1 (no respon
11	0.005840492	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=3/768, ttl=1 (no respon
14	0.005881581	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=4/1024, ttl=2 (no respo
17	0.005921178	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=5/1280, ttl=2 (no respo

Frame 5: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp2s0, id 0 Ethernet II, Src: IntelCor_a5:1e:dc (38:ba:f8:a5:1e:dc), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00) Internet Protocol Version 4, Src: 172.26.12.233, Dst: 193.136.9.240						
0100 = Version: 4 0101 = Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 1500 Identification: 0xd775 (55157) Flags: 0x20, More fragments 0... = Reserved bit: Not set .0.. = Don't fragment: Not set ..1. = More fragments: Set ...0 0000 0000 0000 = Fragment Offset: 0 Time to Live: 1 [Expert Info (Note/Sequence): "Time To Live" only 1] [Time To Live] only 1 [Severity level: Note] [Group: Sequence] Protocol: ICMP (1) Header Checksum: 0x3830 [validation disabled] [Header checksum status: Unverified] Source Address: 172.26.12.233 Destination Address: 193.136.9.240						

Figura 13: Tráfego capturado

Na figura 13, podemos observar a primeira mensagem ICMP. Conseguimos verificar que o pacote foi fragmentado porque a *flag More fragments* contém o valor *Set*. Houve a necessidade de fragmentar o pacote inicial porque o tamanho deste pacote (3607 bytes) é maior do que o MTU (*Maximum Transmission Unit* - neste caso, 1500 bytes), pelo que é vital efetuar fragmentação.

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

A figura 14 mostra o primeiro fragmento do datagrama IP original. Podemos observar que a *flag More fragments* tem o valor *Set*, provando que o datagrama foi fragmentado. Também observamos que *Fragment Offset* contém o valor 0, provando que este fragmento é o primeiro. De forma a calcular o tamanho deste datagrama IP, observamos os valores no campo *total length* nas figuras 14, 15 e 17. Ou seja, o tamanho total será $(1480 + 1480 + 647) + 20 = 3627$ bytes.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.26.12.233	193.137.16.75	DNS	86	Standard query 0x3a4e A marco.uminho.pt OPT
2	0.000328254	172.26.12.233	193.137.16.75	DNS	86	Standard query 0x4598 AAAA marco.uminho.pt OPT
3	0.004498034	193.137.16.75	172.26.12.233	DNS	102	Standard query response 0x3a4e A marco.uminho.pt A 193.136.9.240
4	0.005073429	193.137.16.75	172.26.12.233	DNS	149	Standard query response 0x4598 AAAA marco.uminho.pt SOA dns
5	0.005729937	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no response)
6	0.005763283	172.26.12.233	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=d775)
7	0.005772812	172.26.12.233	193.136.9.240	IPv4	661	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=d775)
8	0.005799295	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=2/512, ttl=1 (no response)
9	0.005810329	172.26.12.233	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=d776)

Frame 5: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp2s0, id 0
Ethernet II, Src: IntelCor_a5:1e:dc (38:ba:f8:a5:1e:dc), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.12.233, Dst: 193.136.9.240
0100 = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0xd775 (55157)
Flags: 0x20, More fragments
0... = Reserved bit: Not set
.0.. = Don't fragment: Not set
..1. = More fragments: Set
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 1
[Expert Info (Note/Sequence): "Time To Live" only 1]
[Time To Live] only 1]
[Severity level: Note]
[Group: Sequence]
Protocol: ICMP (1)

Figura 14: Tráfego capturado para averiguar o processo de fragmentação (Primeiro fragmento)

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

A figura 15 mostra o segundo fragmento do datagrama IP original. Sabemos que não se trata do 1º fragmento uma vez que o valor de *Fragment Offset* contém o valor 1480 e sabemos que existem mais fragmentos porque a *flag More fragments* contém o valor *Set*.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.26.12.233	193.137.16.75	DNS	86	Standard query 0x3a4e A marco.uminho.pt OPT
2	0.000328254	172.26.12.233	193.137.16.75	DNS	86	Standard query 0x4598 AAAA marco.uminho.pt OPT
3	0.004498034	193.137.16.75	172.26.12.233	DNS	102	Standard query response 0x3a4e A marco.uminho.pt A 193.136.149
4	0.005073429	193.137.16.75	172.26.12.233	DNS	149	Standard query response 0x4598 AAAA marco.uminho.pt S0A dns
5	0.005729937	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no respon
6	0.005763283	172.26.12.233	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=d775)
7	0.005772812	172.26.12.233	193.136.9.240	IPv4	661	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=d775)
8	0.005799295	172.26.12.233	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0004, seq=2/512, ttl=1 (no respon
9	0.005810320	172.26.12.233	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=d776)

Frame 6: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp2s0, id 0
 Ethernet II, Src: IntelCor_a5:1e:dc (38:ba:f8:a5:1e:dc), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
 Internet Protocol Version 4, Src: 172.26.12.233, Dst: 193.136.9.240
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 1500
 Identification: 0xd775 (55157)
 Flags: 0x20, More fragments
 0... = Reserved bit: Not set
 .0... = Don't fragment: Not set
 ..1. = More fragments: Set
 ...0 0101 1100 1000 = Fragment Offset: 1480
 Time to Live: 1
 [Expert Info (Note/Sequence): "Time To Live" only 1]
 [Time To Live] only 1
 [Severity level: Note]
 [Group: Sequence]
 Protocol: ICMP (1)

Figura 15: Tráfego capturado para averiguar o processo de fragmentação (Segundo fragmento)

3.4 Estime teoricamente o número de fragmentos gerados a partir do datagrama IP original e o número de bytes transportados no último fragmento desse datagrama. Compare os dois valores estimados com os obtidos através do wireshark.

O esquema da figura 15 tem o intuito de mostrar o processo da fragmentação neste caso. Sabendo que o MTU é de 1500 bytes e que o tamanho do datagrama é de 3607 bytes, a divisão dos fragmentos é a demonstrada na figura - existirão 3 fragmentos. Quanto ao último fragmento, temos de ter a atenção de que todos os fragmentos terão de ter cabeçalho, pelo que 20 bytes de cabeçalho IPv4 serão necessários. Agora, de forma a calcular os outros 647 bytes, teremos que fazer o cálculo $3607 - 1480 - 1480$ bytes, surgindo os 647 bytes que estão na figura.

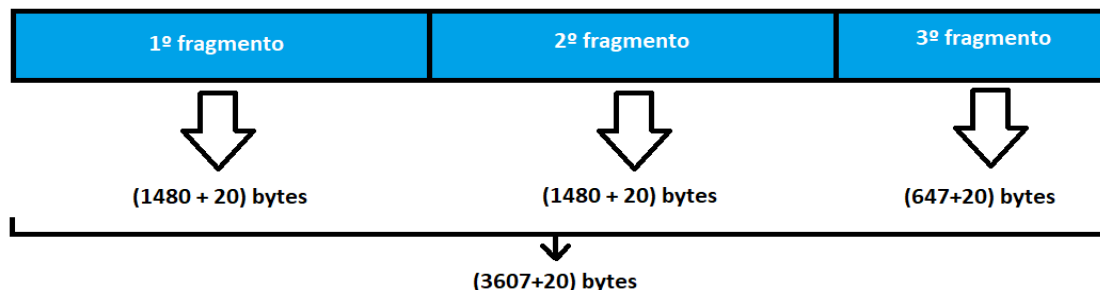


Figura 16: Esquema do cálculo do tamanho total do datagrama sob fragmentação

Tal como já visto neste relatório, o primeiro fragmento consta na figura 14; o segundo fragmento consta na figura 15; e o terceiro e último fragmento consta na figura 17.

No.	Time	Source	Destination	Protocol	Length	Info
7	0.005772812	172.26.12.233	193.136.9.240	IPv4	661	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=d775)
Frame 7: 661 bytes on wire (5288 bits), 661 bytes captured (5288 bits) on interface wlp2s0, id 0 Ethernet II, Src: IntelCor_a5:1e:dc (38:ba:f8:a5:1e:dc), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00) Internet Protocol Version 4, Src: 172.26.12.233, Dst: 193.136.9.240 0100 = Version: 4 0101 = Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) 0000 00.. = Differentiated Services Codepoint: Default (0)00 = Explicit Congestion Notification: Not ECN-Capable Transport (0) Total Length: 647 Identification: 0xd775 (55157) Flags: 0x01 0... = Reserved bit: Not set .0.. = Don't fragment: Not set ..0. = More fragments: Not set ...0 1011 1001 0000 = Fragment Offset: 2960 Time to Live: 1 Protocol: ICMP (1) Header Checksum: 0x5a13 [validation disabled] [Header checksum status: Unverified] Source Address: 172.26.12.233 Destination Address: 193.136.9.240 Data (627 bytes)						

Figura 17: Tráfego capturado para averiguar o processo de fragmentação (Terceiro fragmento)

Podemos observar que os valores do campo *total length* coincidem com os valores da figura 16, assim como número de fragmentos utilizados no processo de fragmentação.

3.5 Como se deteta o último fragmento correspondente ao datagrama original? Estabeleça um filtro no Wireshark que permita listar o último fragmento do primeiro datagrama IP segmentado.

Na figura 18, observamos o filtro no Wireshark para listar o último fragmento do primeiro datagrama IP segmentado. Para tal, sabemos que a flag *more fragments* terá de ter o valor 0, indicando que não existem mais fragmentos depois deste e o *fragment offset* terá de ser superior a 0, indicando que existem mais do que 1 fragmento do mesmo datagrama IP segmentado. Além disso, teremos de filtrar os fragmentos através do seu campo *identification* - que contém o mesmo valor que o datagrama inicial.

ip.flags.mf == 0 && ip.frag_offset > 0 && ip.id == 0xd775						
No.	Time	Source	Destination	Protocol	Length	Info
7	0.005772812	172.26.12.233	193.136.9.240	IPv4	661	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=d775)

Figura 18: Filtro no Wireshark

3.6 Identifique o equipamento onde o datagrama IP original é reconstruído a partir dos fragmentos. A reconstrução poderia ter ocorrido noutro equipamento diferente do identificado? Porquê?

O datagrama IP original é reconstruído no *host* destino, pois é ele o recetor do datagrama IP cujo endereço consta no campo IP Destination e, se ocorrer a reconstrução dos fragmentos noutro local, poderá ocorrer, mais tarde, o processo de fragmentação novamente, levando a um overhead desnecessário na rede.

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

Os campos alterados são a identificação (*identification*), as flags e o offset. Na reconstrução do datagrama original, o recetor (*host destino*) usa o campo de identificação para apontar os

fragmentos pertencentes ao mesmo pacote original. As flags são usadas para identificar o último fragmento. O offset permite colocar cada fragmento na devida posição do datagrama original.

3.8 Por que razão apenas o primeiro fragmento de cada pacote é identificado como sendo um pacote ICMP?

O cabeçalho ICMP é usado para enviar mensagens de erro entre dispositivos da rede. As mensagens são enviadas em pacotes independentes que não necessitam de ser fragmentados. Se um pacote ICMP for fragmentado, apenas o primeiro fragmento conterá o cabeçalho ICMP original. Os outros fragmentos irão conter apenas o cabeçalho IP original.

3.9 Com que valor é o tamanho do datagrama comparado a fim de se determinar se este deve ser fragmentado? Quais seriam os efeitos na rede ao aumentar/diminuir este valor?

Para determinar se o datagrama deve ou não ser fragmentado, o seu tamanho deve ser comparado com o MTU (Maximum Transmission Unit). Este valor corresponde ao tamanho máximo do pacote que pode ser transmitido numa única transmissão, sem que haja a necessidade de o mesmo ser fragmentado. Aumentar o MTU resulta na existência de pacotes maiores, o que implica menos fragmentação. Isto poderia permitir uma transmissão mais eficiente e uma melhor utilização da largura de banda disponível na rede. A latência poderia ainda ser reduzida devido ao menor tempo de processamento. No entanto, um maior MTU poderá aumentar a probabilidade de perda de pacotes, visto que pacotes maiores têm mais probabilidade de encontrar erros na transmissão. Na diminuição do MTU verifica-se o inverso ao dito anteriormente.

3.10 Sabendo que no comando ping a opção -f (Windows), -M do (Linux) ou -D (Mac) 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, Mac), determine o valor máximo de SIZE sem que ocorra fragmentação do pacote? Justifique o valor obtido.

```
user@vivobook [~] ○
> ping -M do -s 1472 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 1472(1500) bytes of data.
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=1 ttl=61 time=4.45 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=2 ttl=61 time=6.19 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=3 ttl=61 time=6.03 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=4 ttl=61 time=5.39 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=5 ttl=61 time=5.56 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=6 ttl=61 time=6.80 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=7 ttl=61 time=6.24 ms
^C
--- marco.uminho.pt ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6010ms
rtt min/avg/max/mdev = 4.449/5.807/6.802/0.702 ms
user@vivobook [~] ○
> ping -M do -s 1473 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 1473(1501) bytes of data.
ping: local error: message too long, mtu=1500
ping: local error: message too long, mtu=1500
ping: local error: message too long, mtu=1500
ping: local error: message too long, mtu=1500
ping: local error: message too long, mtu=1500
ping: local error: message too long, mtu=1500
^C
--- marco.uminho.pt ping statistics ---
6 packets transmitted, 0 received, +6 errors, 100% packet loss, time 5126ms
```

Figura 19: Execução do comando ping

Através da figura 19, podemos observar que o MTU da rede em questão é de 1500 bytes. Tendo em conta os tamanhos dos cabeçalhos IP e ICMP (20 e 8, respetivamente), sabemos que o valor máximo para o SIZE terá de ser $1500 - 20 - 8 = 1472$ bytes. Para efeitos de teste, atribuímos 1473 à variável SIZE e reparamos que não foi executado o comando ping (o pacote foi descartado).

Parte II: Endereçamento e Encaminhamento IP

4

- 4.1 Averigue, através do comando ping, que AfonsoHenriques tem efetivamente conectividade com o servidor Finanças e com os servidores da CDN.

```
root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# ping 192.168.0.250
PING 192.168.0.250 (192.168.0.250) 56(84) bytes of data.
64 bytes from 192.168.0.250: icmp_seq=1 ttl=61 time=0.209 ms
64 bytes from 192.168.0.250: icmp_seq=2 ttl=61 time=0.101 ms
64 bytes from 192.168.0.250: icmp_seq=3 ttl=61 time=0.066 ms
64 bytes from 192.168.0.250: icmp_seq=4 ttl=61 time=0.061 ms
^C
--- 192.168.0.250 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3050ms
rtt min/avg/max/mdev = 0.061/0.109/0.209/0.059 ms
```

Figura 20: traceroute AfonsoHenriques → Finanças

```
root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# ping 192.168.0.203
PING 192.168.0.203 (192.168.0.203) 56(84) bytes of data.
64 bytes from 192.168.0.203: icmp_seq=1 ttl=55 time=0.132 ms
64 bytes from 192.168.0.203: icmp_seq=2 ttl=55 time=0.099 ms
64 bytes from 192.168.0.203: icmp_seq=3 ttl=55 time=0.237 ms
64 bytes from 192.168.0.203: icmp_seq=4 ttl=55 time=0.146 ms
^C
--- 192.168.0.203 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3083ms
rtt min/avg/max/mdev = 0.099/0.153/0.237/0.051 ms
```

Figura 21: traceroute AfonsoHenriques → Netflix

```
root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# ping 192.168.0.202
PING 192.168.0.202 (192.168.0.202) 56(84) bytes of data.
64 bytes from 192.168.0.202: icmp_seq=1 ttl=55 time=0.195 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.126 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.117 ms
64 bytes from 192.168.0.202: icmp_seq=4 ttl=55 time=0.140 ms
^C
--- 192.168.0.202 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.117/0.144/0.195/0.030 ms
```

Figura 22: traceroute AfonsoHenriques → Youtube

```
root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# ping 192.168.0.250
PING 192.168.0.250 (192.168.0.250) 56(84) bytes of data.
64 bytes from 192.168.0.250: icmp_seq=1 ttl=61 time=0.209 ms
64 bytes from 192.168.0.250: icmp_seq=2 ttl=61 time=0.101 ms
64 bytes from 192.168.0.250: icmp_seq=3 ttl=61 time=0.066 ms
64 bytes from 192.168.0.250: icmp_seq=4 ttl=61 time=0.061 ms
^C
--- 192.168.0.250 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3050ms
rtt min/avg/max/mdev = 0.061/0.109/0.209/0.059 ms
```

Figura 23: traceroute Afonso Henriques → HBO


```

root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# ping 192.168.0.204
PING 192.168.0.204 (192.168.0.204) 56(84) bytes of data.
64 bytes from 192.168.0.204: icmp_seq=1 ttl=55 time=0.163 ms
64 bytes from 192.168.0.204: icmp_seq=2 ttl=55 time=0.104 ms
64 bytes from 192.168.0.204: icmp_seq=3 ttl=55 time=0.103 ms
64 bytes from 192.168.0.204: icmp_seq=4 ttl=55 time=0.106 ms
^C
--- 192.168.0.204 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3072ms
rtt min/avg/max/mdev = 0.103/0.119/0.163/0.025 ms

```

Figura 24: traceroute AfonsoHenriques → iTunes

```

root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# ping 192.168.0.218
PING 192.168.0.218 (192.168.0.218) 56(84) bytes of data.
64 bytes from 192.168.0.218: icmp_seq=1 ttl=55 time=0.105 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=55 time=0.219 ms
64 bytes from 192.168.0.218: icmp_seq=3 ttl=55 time=0.159 ms
64 bytes from 192.168.0.218: icmp_seq=4 ttl=55 time=0.110 ms
^C
--- 192.168.0.218 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3059ms
rtt min/avg/max/mdev = 0.105/0.148/0.219/0.045 ms

```

Figura 25: traceroute AfonsoHenriques → Spotify

Através das figuras acima, podemos observar que, a título de ilustração, foram recebidos 4 pacotes de prova dos 4 transmitidos, pelo que podemos concluir que existe conectividade entre AfonsoHenriques e o servidor das Finanças, e AfonsoHenriques com os servidores da CDN.

4.2 Recorrendo ao comando `netstat -rn`, analise as tabelas de encaminhamento dos dispositivos AfonsoHenriques e Teresa. Existe algum problema com as suas entradas? Identifique e descreva a utilidade de cada uma das entradas destes dois hosts.

```

root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 192.168.0.225 0.0.0.0 UG 0 0 0 eth0
192.168.0.224 0.0.0.0 255.255.255.248 U 0 0 0 eth0

```

Figura 26: Tabela de routing de AfonsoHenriques

```

root@Teresa:/tmp/pycore.40445/Teresa.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 192.168.0.193 0.0.0.0 UG 0 0 0 eth0
192.168.0.192 0.0.0.0 255.255.255.248 U 0 0 0 eth0
root@Teresa:/tmp/pycore.40445/Teresa.conf# █

```

Figura 27: Tabela de routing de Teresa

Nas figuras 26 e 27, podemos observar as tabelas de routing de AfonsoHenriques e Teresa, que não apresentam quaisquer problemas.

Estas contêm as seguintes colunas (que são mais importantes para o exercício):

- **Destination** é um campo que indica para qual rede ou host uma rota é aplicada;
- **Gateway** é o endereço IP do próximo dispositivo de rede que deve ser usado para encaminhar um pacote de dados;
- **Genmask** é um campo que especifica a máscara de subrede para uma determinada entrada na tabela de routing. A máscara de subrede é usada para determinar qual a parte do endereço IP do destino deve ser considerada a rede e qual a parte deve ser considerada o host.

Também podemos identificar duas entradas que são análogas em ambas as tabelas:

- a primeira refere-se à ligação entre a rede do host em questão e um destino pertencente a uma sub-rede externa (representado como 0.0.0.0), sendo o gateway 192.168.0.255 ou 192.168.0.193 - consoante seja AfonsoHenriques ou Teresa, respetivamente (sinalizada pela flag UG), que é referente ao IP da interface ativa do router que vai depois encaminhar o pacote para outro router de forma a fazê-lo chegar ao seu destino;
- a segunda entrada é referente a ligação para um dispositivo da subrede, que é sinalizado com a flag U, sendo que o gateway é 0.0.0.0 (ligação direta), e a genmask é 255.255.255.248 (que é referente a máscara de rede /29).

4.3 Utilize o Wireshark para investigar o comportamento dos routers do core da rede (n1 a n6) quando tenta estabelecer comunicação entre os hosts AfonsoHenriques e Teresa. Indique que dispositivo(s) não permite(m) o encaminhamento correto dos pacotes. Seguidamente, avalie e explique a(s) causa(s) do funcionamento incorreto do dispositivo.

Utilize o comando `ip route add/del` para adicionar as rotas necessárias ou remover rotas incorretas. Verifique a sintaxe completa do comando a usar com `man ip-route` ou `man route`. Poderá também utilizar o comando `traceroute` para se certificar do caminho nó a nó. Considere a alínea resolvida assim que houver tráfego a chegar ao ISP CondadOnline.

```
root@AfonsoHenriques:/tmp/pycore.40445/AfonsoHenriques.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225)  0.335 ms  0.011 ms  0.007 ms
 2 172.16.143.1 (172.16.143.1)  0.025 ms  0.011 ms  0.011 ms
 3 10.0.0.29 (10.0.0.29)  0.025 ms !N  0.014 ms !N *
```

Figura 28: Traceroute AfonsoHenriques → Teresa

Analisando a figura 28 relativamente ao comando `traceroute` da ligação entre AfonsoHenriques e Teresa, pela flag `!N` e pela falta de saltos de que deviam existir de acordo com a topologia, concluímos que é perdido o contacto em n5.

```
root@n5:/tmp/pycore.40445/n5.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
connect: Network is unreachable
root@n5:/tmp/pycore.40445/n5.conf# █
```

Figura 29: Traceroute n5 → Teresa

Para confirmar que realmente é perdido o contacto em n5, é realizado o `traceroute` entre n5 e Teresa. Na figura 29, vemos a mensagem "Network is unreachable", pelo que concluímos que realmente não existe rota.

```

root@n3:/tmp/pycore.40445/n3.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1  10.0.0.5 (10.0.0.5)  0.068 ms  0.007 ms  0.005 ms
 2  10.0.0.1 (10.0.0.1)  0.026 ms  0.007 ms  0.007 ms
 3  172.16.142.2 (172.16.142.2)  0.030 ms  0.010 ms  0.010 ms
 4  192.168.0.194 (192.168.0.194)  0.031 ms  0.154 ms  0.088 ms

```

Figura 30: Traceroute n3 → Teresa

Com o objetivo de verificar se existe algum problema depois de n5, feito o traceroute desde n3 até Teresa. Pela figura 30 percebemos que existe uma rota entre os dois, logo o problema está antes de n3.

```

root@n4:/tmp/pycore.40445/n4.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1  10.0.0.17 (10.0.0.17)  0.036 ms  0.004 ms  0.003 ms
 2  10.0.0.5 (10.0.0.5)  0.015 ms  0.006 ms  0.005 ms
 3  10.0.0.1 (10.0.0.1)  0.015 ms  0.007 ms  0.008 ms
 4  172.16.142.2 (172.16.142.2)  0.017 ms  0.009 ms  0.009 ms
 5  192.168.0.194 (192.168.0.194)  0.022 ms  0.011 ms  0.011 ms
root@n4:/tmp/pycore.40445/n4.conf# █

```

Figura 31: Traceroute n4 → Teresa

Foi feita uma execução do comando traceroute desde n4 até Teresa para identificar a rota onde existe problemas. Através da figura 31 percebemos que foi estabelecido contacto. Concluimos então que o problema está na rota n5 → n2 → n1 → n3.

Assim, começaremos por ver a tabela de encaminhamento do router n5 na figura 32.

```

Kernel IP routing table
Destination    Gateway         Genmask         Flags        MSS Window  irtt Iface
10.0.0.0       10.0.0.25      255.255.255.252 UG           0 0        0 eth1
10.0.0.4       10.0.0.25      255.255.255.252 UG           0 0        0 eth1
10.0.0.8       10.0.0.25      255.255.255.252 UG           0 0        0 eth1
10.0.0.12      10.0.0.25      255.255.255.252 UG           0 0        0 eth1
10.0.0.16      10.0.0.25      255.255.255.252 UG           0 0        0 eth1
10.0.0.20      10.0.0.25      255.255.255.252 UG           0 0        0 eth1
10.0.0.24      0.0.0.0        255.255.255.252 U            0 0        0 eth1
10.0.0.28      0.0.0.0        255.255.255.252 U            0 0        0 eth0
172.0.0.0      10.0.0.30      255.0.0.0       UG           0 0        0 eth0
172.16.142.0   10.0.0.25      255.255.255.248 UG           0 0        0 eth1
172.16.143.0   10.0.0.30      255.255.255.252 UG           0 0        0 eth0
172.16.143.0   10.0.0.30      255.255.255.248 UG           0 0        0 eth0
172.16.143.4   10.0.0.30      255.255.255.252 UG           0 0        0 eth0
192.142.0.4    10.0.0.25      255.255.255.252 UG           0 0        0 eth1
192.168.0.200  10.0.0.25      255.255.255.248 UG           0 0        0 eth1
192.168.0.208  10.0.0.25      255.255.255.248 UG           0 0        0 eth1
192.168.0.216  10.0.0.25      255.255.255.248 UG           0 0        0 eth1
192.168.0.224  10.0.0.30      255.255.255.248 UG           0 0        0 eth0
192.168.0.232  10.0.0.30      255.255.255.248 UG           0 0        0 eth0
192.168.0.240  10.0.0.30      255.255.255.248 UG           0 0        0 eth0
192.168.0.248  10.0.0.30      255.255.255.248 UG           0 0        0 eth0
root@n5:/tmp/pycore.32913/n5.conf# █

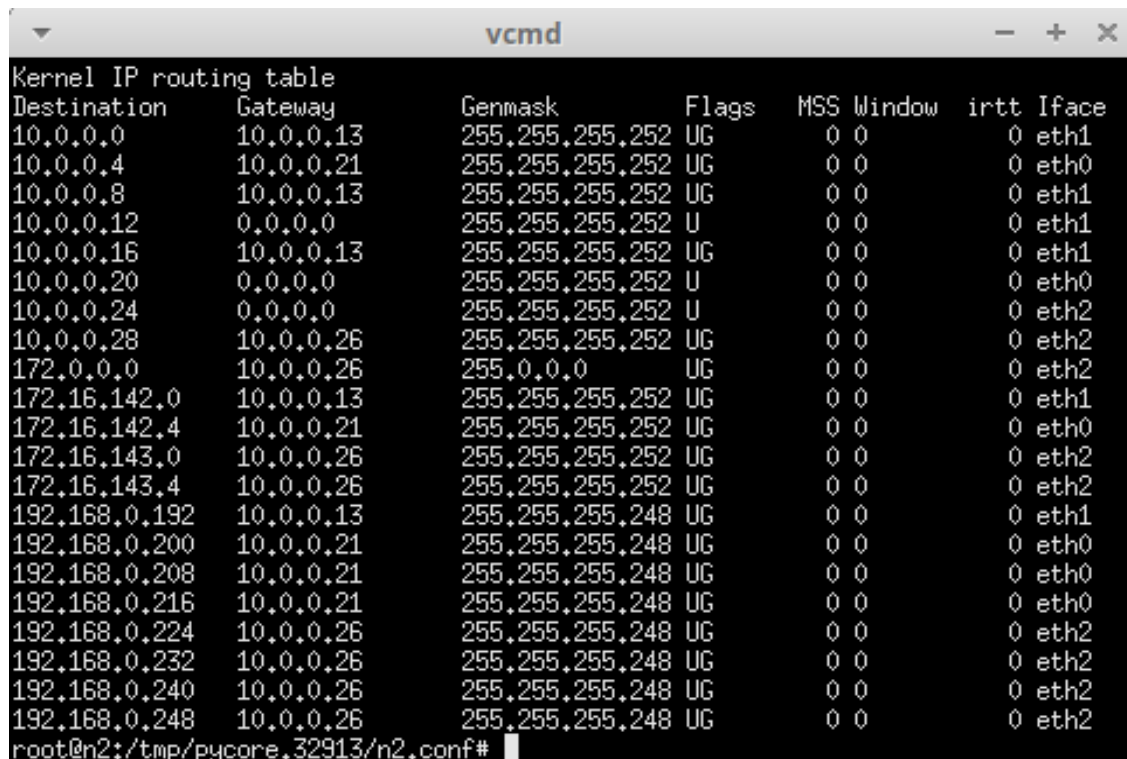
```

Figura 32: Tabela de routing de n5

De seguida, iremos efetuar algumas operações:

- `route del -net 172.16.143.0 netmask 255.255.255.248 gw 10.0.0.30` - entrada que já estava na tabela e que contém a máscara errada (tinha /29 e deve apenas estar /30);
- `route del -net 172.16.142.0 netmask 255.255.255.248 gw 10.0.0.25` e `route add -net 172.16.142.0 netmask 255.255.255.252 gw 10.0.0.25` - alteração para a máscara correta
- `route del -net 192.142.0.4 netmask 255.255.255.252 gw 10.0.0.25` - *ip destination* não existente na topologia
- `route add -net 172.16.142.4 netmask 255.255.255.252 gw 10.0.0.25` - rota para a interface do RACDN para o polo CDN
- `route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.25` - rota para a subrede de Galiza

Já no router n2, efetuamos a operação `route del -net 192.168.0.194 netmask 255.255.255.254 gw 10.0.0.25` - entrada desnecessária pelo que basta uma entrada para a subrede Galiza. Além disso, tem a máscara errada (deveria estar /29 e está /31) (figura 33)



Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
10.0.0.0	10.0.0.13	255.255.255.252	UG	0	0	0	eth1
10.0.0.4	10.0.0.21	255.255.255.252	UG	0	0	0	eth0
10.0.0.8	10.0.0.13	255.255.255.252	UG	0	0	0	eth1
10.0.0.12	0.0.0.0	255.255.255.252	U	0	0	0	eth1
10.0.0.16	10.0.0.13	255.255.255.252	UG	0	0	0	eth1
10.0.0.20	0.0.0.0	255.255.255.252	U	0	0	0	eth0
10.0.0.24	0.0.0.0	255.255.255.252	U	0	0	0	eth2
10.0.0.28	10.0.0.26	255.255.255.252	UG	0	0	0	eth2
172.0.0.0	10.0.0.26	255.0.0.0	UG	0	0	0	eth2
172.16.142.0	10.0.0.13	255.255.255.252	UG	0	0	0	eth1
172.16.142.4	10.0.0.21	255.255.255.252	UG	0	0	0	eth0
172.16.143.0	10.0.0.26	255.255.255.252	UG	0	0	0	eth2
172.16.143.4	10.0.0.26	255.255.255.252	UG	0	0	0	eth2
192.168.0.192	10.0.0.13	255.255.255.248	UG	0	0	0	eth1
192.168.0.200	10.0.0.21	255.255.255.248	UG	0	0	0	eth0
192.168.0.208	10.0.0.21	255.255.255.248	UG	0	0	0	eth0
192.168.0.216	10.0.0.21	255.255.255.248	UG	0	0	0	eth0
192.168.0.224	10.0.0.26	255.255.255.248	UG	0	0	0	eth2
192.168.0.232	10.0.0.26	255.255.255.248	UG	0	0	0	eth2
192.168.0.240	10.0.0.26	255.255.255.248	UG	0	0	0	eth2
192.168.0.248	10.0.0.26	255.255.255.248	UG	0	0	0	eth2

Figura 33: Tabela de routing de n2

De seguida, experimentamos efetuar um comando `tracert` de AfonsoHenriques até CondaOnline e verificamos que os pacotes já lá chegam, pelo que podemos considerar que esta alínea já está resolvida.


```

root@n5:/tmp/pycore.46311/n5.conf# traceroute 192.168.0.193
traceroute to 192.168.0.193 (192.168.0.193), 30 hops max, 60 byte packets
 1  10.0.0.25 (10.0.0.25)  0.031 ms  0.005 ms  0.004 ms
 2  10.0.0.13 (10.0.0.13)  0.017 ms  0.009 ms  0.007 ms
 3  10.0.0.25 (10.0.0.25)  0.007 ms  0.007 ms  0.007 ms
 4  10.0.0.13 (10.0.0.13)  0.008 ms  0.009 ms  0.009 ms
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * 10.0.0.13 (10.0.0.13)  0.076 ms  0.027 ms
11  10.0.0.25 (10.0.0.25)  0.028 ms  0.024 ms  0.023 ms
12  10.0.0.13 (10.0.0.13)  0.026 ms  0.028 ms  0.027 ms
13  10.0.0.25 (10.0.0.25)  0.026 ms  0.027 ms *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * 10.0.0.25 (10.0.0.25)  0.077 ms  0.032 ms
20  10.0.0.13 (10.0.0.13)  0.037 ms  0.034 ms  0.033 ms
21  10.0.0.25 (10.0.0.25)  0.033 ms  0.033 ms  0.035 ms
22  10.0.0.13 (10.0.0.13)  0.037 ms  0.036 ms *^C
root@n5:/tmp/pycore.46311/n5.conf#

```

Figura 36: Loop com n2 e n1

Neste ponto, suspeitamos que o problema poderá estar no router n1 e, observando a sua tabela de routing (figura 37), reparávamos que todo o tráfego com o destino localizado nos polos mais a direita na figura estava a ser reencaminhado para o router n2, que por sua vez estava a ser reencaminhado para n1, e assim sucessivamente. Ora, efetuando os comandos para passar a reencaminhar o tráfego para n3 (route del -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.14 e route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.9).

```

root@n1:/tmp/pycore.46311/n1.conf# netstat -nr
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.0.9        255.255.255.252 UG          0 0         0 eth0
10.0.0.4        10.0.0.9        255.255.255.252 UG          0 0         0 eth0
10.0.0.8        0.0.0.0         255.255.255.252 U          0 0         0 eth0
10.0.0.12       0.0.0.0         255.255.255.252 U          0 0         0 eth1
10.0.0.16       10.0.0.9        255.255.255.252 UG          0 0         0 eth0
10.0.0.20       10.0.0.14       255.255.255.252 UG          0 0         0 eth1
10.0.0.24       10.0.0.14       255.255.255.252 UG          0 0         0 eth1
10.0.0.28       10.0.0.14       255.255.255.252 UG          0 0         0 eth1
172.0.0.0       10.0.0.14       255.0.0.0       UG          0 0         0 eth1
172.16.142.0    10.0.0.9        255.255.255.252 UG          0 0         0 eth0
172.16.142.4    10.0.0.9        255.255.255.252 UG          0 0         0 eth0
172.16.143.0    10.0.0.14       255.255.255.252 UG          0 0         0 eth1
172.16.143.4    10.0.0.14       255.255.255.252 UG          0 0         0 eth1
192.168.0.192   10.0.0.14       255.255.255.248 UG          0 0         0 eth1
192.168.0.200   10.0.0.9        255.255.255.248 UG          0 0         0 eth0
192.168.0.208   10.0.0.9        255.255.255.248 UG          0 0         0 eth0
192.168.0.216   10.0.0.9        255.255.255.248 UG          0 0         0 eth0
192.168.0.224   10.0.0.14       255.255.255.248 UG          0 0         0 eth1
192.168.0.232   10.0.0.14       255.255.255.248 UG          0 0         0 eth1
192.168.0.240   10.0.0.14       255.255.255.248 UG          0 0         0 eth1
192.168.0.248   10.0.0.14       255.255.255.248 UG          0 0         0 eth1
root@n1:/tmp/pycore.46311/n1.conf# route del -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.14
root@n1:/tmp/pycore.46311/n1.conf# route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.9
root@n1:/tmp/pycore.46311/n1.conf#

```

Figura 37: Tabela de routing de n1

Nesta fase, n1 já consegue ter ligação a Teresa (figura 38).

```

root@n1:/tmp/pycore.46311/n1.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1  10.0.0.9 (10.0.0.9)  0.031 ms  0.005 ms  0.004 ms
 2  10.0.0.5 (10.0.0.5)  0.016 ms  0.006 ms  0.005 ms
 3  10.0.0.1 (10.0.0.1)  0.014 ms  0.007 ms  0.018 ms
 4  172.16.142.2 (172.16.142.2)  0.019 ms  0.009 ms  0.009 ms
 5  192.168.0.194 (192.168.0.194)  0.018 ms  0.010 ms  0.011 ms
root@n1:/tmp/pycore.46311/n1.conf# █

```

Figura 38: Traceroute n1 → Teresa

No entanto, AfonsoHenriques ainda não tem conectividade para a Teresa, tal como observado em baixo.

```

<3199/AfonsoHenriques.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.068 ms  0.010 ms  0.004 ms
 2  172.16.143.1 (172.16.143.1)  0.017 ms  0.007 ms  0.005 ms
 3  10.0.0.29 (10.0.0.29)  0.017 ms  0.008 ms  0.008 ms
 4  10.0.0.25 (10.0.0.25)  0.024 ms  0.009 ms  0.008 ms
 5  10.0.0.13 (10.0.0.13)  0.024 ms  0.013 ms  0.013 ms
 6  10.0.0.17 (10.0.0.17)  0.062 ms  0.028 ms  0.014 ms
 7  10.0.0.5 (10.0.0.5)  0.029 ms  0.015 ms  0.014 ms
 8  10.0.0.1 (10.0.0.1)  0.026 ms  0.024 ms  0.016 ms
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  *^C
root@AfonsoHenriques:/tmp/pycore.43199/AfonsoHenriques.conf# █

```

Figura 39: Traceroute AfonsoHenriques → Teresa

Com isto, depois de efetuarmos traceroute de todos os polos para a Teresa com sucesso, supomos que o problema seria no router de acesso RAGaliza. Aqui reparamos que este não conhecia a rede para o polo Condado Portucalense, procedendo a adição de uma nova entrada através do comando `route add -net 192.168.0.224 netmask 255.255.255.248 gw 192.16.142.1`.

Depois da execução deste comando, procedemos à execução de um novo traceroute do AfonsoHenriques até Teresa e já existe conectividade.

```

<3199/AfonsoHenriques.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.179 ms 0.012 ms 0.007 ms
 2 172.16.143.1 (172.16.143.1) 0.024 ms 0.011 ms 0.010 ms
 3 10.0.0.29 (10.0.0.29) 0.028 ms 0.014 ms 0.014 ms
 4 10.0.0.25 (10.0.0.25) 0.032 ms 0.017 ms 0.018 ms
 5 10.0.0.13 (10.0.0.13) 0.044 ms 0.020 ms 0.039 ms
 6 10.0.0.17 (10.0.0.17) 0.071 ms 0.033 ms 0.014 ms
 7 10.0.0.5 (10.0.0.5) 0.029 ms 0.016 ms 0.015 ms
 8 10.0.0.1 (10.0.0.1) 0.026 ms 0.045 ms 0.021 ms
 9 172.16.142.2 (172.16.142.2) 0.030 ms 0.019 ms 0.020 ms
10 192.168.0.194 (192.168.0.194) 0.035 ms 0.021 ms 0.021 ms
root@AfonsoHenriques:/tmp/pycore.43199/AfonsoHenriques.conf#

```

Figura 40: Traceroute AfonsoHenriques → Teresa (com ligação)

(b) As rotas dos pacotes ICMP echo reply são as mesmas, mas em sentido inverso, que as rotas dos pacotes ICMP echo request enviados entre AfonsoHenriques e Teresa? (Sugestão: analise as rotas nos dois sentidos com o traceroute). Mostre graficamente a rota seguida nos dois sentidos por esses pacotes ICMP. Nas figuras abaixo, podemos observar a execução do comando traceroute do AfonsoHenriques até Teresa e outra execução da Teresa até AfonsoHenriques.

```

<3199/AfonsoHenriques.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.179 ms 0.012 ms 0.007 ms
 2 172.16.143.1 (172.16.143.1) 0.024 ms 0.011 ms 0.010 ms
 3 10.0.0.29 (10.0.0.29) 0.028 ms 0.014 ms 0.014 ms
 4 10.0.0.25 (10.0.0.25) 0.032 ms 0.017 ms 0.018 ms
 5 10.0.0.13 (10.0.0.13) 0.044 ms 0.020 ms 0.039 ms
 6 10.0.0.17 (10.0.0.17) 0.071 ms 0.033 ms 0.014 ms
 7 10.0.0.5 (10.0.0.5) 0.029 ms 0.016 ms 0.015 ms
 8 10.0.0.1 (10.0.0.1) 0.026 ms 0.045 ms 0.021 ms
 9 172.16.142.2 (172.16.142.2) 0.030 ms 0.019 ms 0.020 ms
10 192.168.0.194 (192.168.0.194) 0.035 ms 0.021 ms 0.021 ms
root@AfonsoHenriques:/tmp/pycore.43199/AfonsoHenriques.conf#

```

Figura 41: Traceroute AfonsoHenriques → Teresa

```

root@Teresa:/tmp/pycore.43199/Teresa.conf# traceroute 192.168.0.226
traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
 1 192.168.0.193 (192.168.0.193) 0.060 ms 0.005 ms 0.004 ms
 2 172.16.142.1 (172.16.142.1) 0.034 ms 0.006 ms 0.006 ms
 3 10.0.0.2 (10.0.0.2) 0.020 ms 0.007 ms 0.007 ms
 4 10.0.0.6 (10.0.0.6) 0.019 ms 0.009 ms 0.009 ms
 5 10.0.0.18 (10.0.0.18) 0.022 ms 0.011 ms 0.012 ms
 6 10.0.0.14 (10.0.0.14) 0.037 ms 0.083 ms 0.033 ms
 7 10.0.0.26 (10.0.0.26) 0.041 ms 0.015 ms 0.017 ms
 8 10.0.0.30 (10.0.0.30) 0.030 ms 0.018 ms 0.017 ms
 9 172.16.143.2 (172.16.143.2) 0.072 ms 0.023 ms 0.018 ms
10 192.168.0.226 (192.168.0.226) 0.028 ms 0.021 ms 0.020 ms
root@Teresa:/tmp/pycore.43199/Teresa.conf#

```

Figura 42: Traceroute Teresa → AfonsoHenriques

Graficamente, podemos observar as rotas observadas nas figuras acima na figura abaixo - a verde observamos a rota **Teresa → AfonsoHenriques** e a vermelho observamos a rota **AfonsoHenriques → Teresa**:

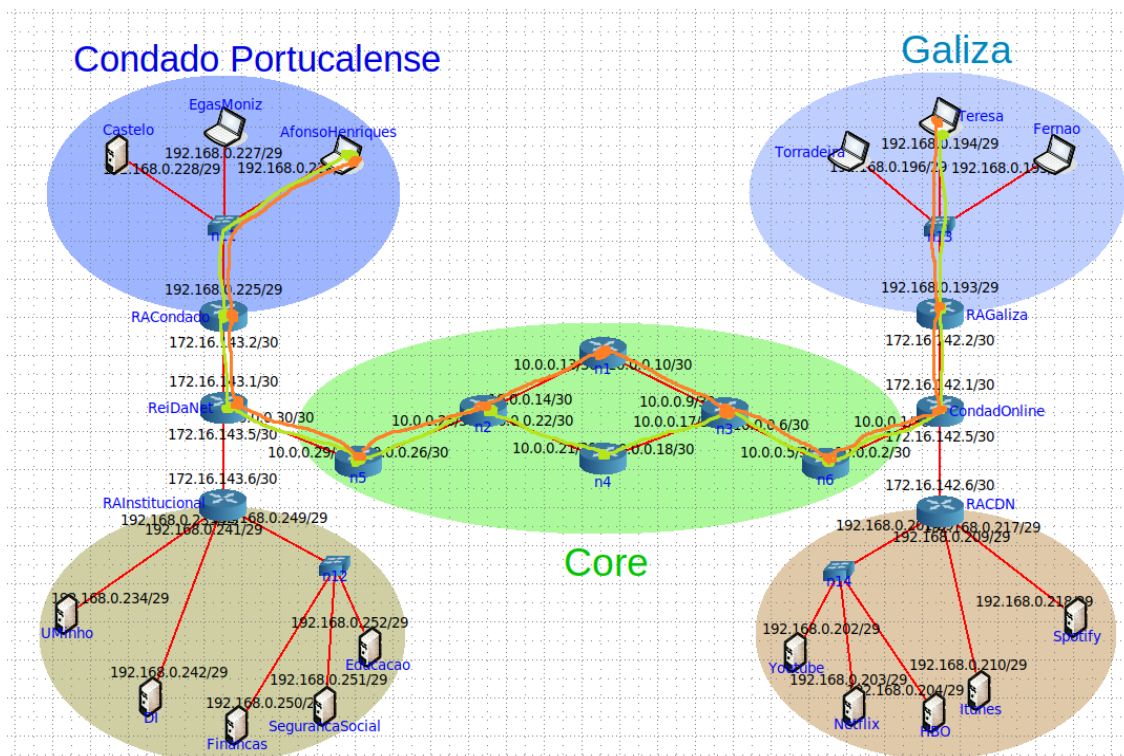


Figura 43: Topologia com as rotas dos comandos traceroute

Analisando a figura acima, podemos concluir que as rotas dos pacotes ICMP echo request enviados entre AfonsoHenriques e Teresa são diferentes nos dois sentidos.

- 4.5** Estando restabelecida a conectividade entre os dois hosts, obtenha a tabela de encaminhamento de n3 e foque-se na seguinte entrada: Existe uma correspondência (match) nesta entrada para pacotes enviados para o polo Galiza? E para CDN? Caso seja essa a entrada utilizada para o encaminhamento, permitirá o funcionamento esperado do dispositivo? Ofereça uma explicação pela qual essa entrada é ou não utilizada.

A entrada que está presente na figura, ou seja, 192.168.0.192 10.0.0.18 255.255.255.240 (...), mostra que o router n3 tem conhecimento da rede 192.168.0.192, cujo gateway/next hop é 10.0.0.18. Por outras palavras, o router n3 tem conhecimento da subrede do polo da Galiza e, para encaminhar pacotes para lá, terá que os enviar para o router n6. No entanto, a máscara que se encontra nesta entrada está errada (deveria ser um /29 e atualmente é /28), pelo que esta entrada não é utilizada, já que o sistema usará a entrada mais específica para determinar a rota, ou seja, a entrada com a máscara mais longa (longest prefix matching). Já para encaminhar pacotes para o polo CDN, o router n3 está impossibilitado de enviar pacotes para lá pois no campo Destination não está nenhum dos endereços IP das 3 subredes que existem no polo CDN (192.168.0.200, 192.168.0.208 e 192.168.0.216). Para poder entregar pacotes para lá, terá de haver uma entrada na tabela de encaminhamento para cada um destes endereços IP.

- 4.6** Os endereços utilizados pelos quatro polos são endereços públicos ou privados? E os utilizados no core da rede/ISPs? Justifique convenientemente.

Os endereços utilizados pelos quatro polos são endereços privados, uma vez que estão na gama dos endereços privados 192.168.0.0 - 192.168.255.255/16 e são utilizados em subredes nos diversos routers de acesso para os diferentes polos. Também os endereços utilizados no core da rede/ISPs também são endereços privados pois inserem-se na gama de endereços 172.16.0.0 - 172.31.255.255 (/12) e 10.0.0.0 - 10.255.255.255 (/8), respetivamente.

- 4.7** Os switches localizados em cada um dos polos têm um endereço IP atribuído? Porquê?

Os switches não têm endereços IP atribuídos, uma vez que apenas opera até ao nível 2 da pilha protocolar (i.e. nível lógico). Para tal, teria de operar na camada seguinte - camada de rede. Outro motivo é o facto deste dispositivo apenas operar sobre endereços físicos - é um endereço IP é de natureza lógica.

5

- 5.1** Não estando satisfeito com a decoração do Castelo, opta por eliminar a sua rota default. Adicione as rotas necessárias para que o Castelo continue a ter acesso a cada um dos três polos. Mostre que a conectividade é restabelecida, assim como a tabela de encaminhamento resultante. Explícite ainda a utilidade de uma rota default.

A rota default é uma rota usada por dispositivos de rede para definir a rota padrão que cada pacote de dados deve seguir quando não há uma rota mais específica definida na tabela de routing.

Quando um dispositivo precisa enviar um pacote de dados para um destino fora da sua rede local, ele usa uma tabela de routing para determinar a melhor rota para enviar o pacote. Se não houver uma rota mais específica definida para o destino, o dispositivo usará a rota padrão ou *default* para enviar o pacote.



```

vcmd
root@Castelo:/tmp/pycore.38147/Castelo.conf# traceroute 192.168.0.228
traceroute to 192.168.0.228 (192.168.0.228), 30 hops max, 60 byte packets
 1 192.168.0.228 (192.168.0.228) 0.027 ms 0.002 ms 0.003 ms
root@Castelo:/tmp/pycore.38147/Castelo.conf# netstat -nr
Kernel IP routing table
Destination      Gateway          Genmask         Flags       MSS Window  irtt Iface
0.0.0.0          192.168.0.225   0.0.0.0         UG          0 0        0 eth0
192.168.0.224    0.0.0.0         255.255.255.248 U           0 0        0 eth0
<147/Castelo.conf# route del -net 0.0.0.0 netmask 0.0.0.0 gw 192.168.0.225
root@Castelo:/tmp/pycore.38147/Castelo.conf# netstat -nr
Kernel IP routing table
Destination      Gateway          Genmask         Flags       MSS Window  irtt Iface
192.168.0.224    0.0.0.0         255.255.255.248 U           0 0        0 eth0
root@Castelo:/tmp/pycore.38147/Castelo.conf# █

```

Figura 44: Eliminação da route default

1. Eliminação da rota com o comando `route del -net 0.0.0.0 netmask 0.0.0.0 gw 192.168.0.225`.



```

root@Castelo:/tmp/pycore.38147/Castelo.conf# traceroute 192.168.0.234
traceroute to 192.168.0.234 (192.168.0.234), 30 hops max, 60 byte packets
connect: Network is unreachable
root@Castelo:/tmp/pycore.38147/Castelo.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
connect: Network is unreachable
root@Castelo:/tmp/pycore.38147/Castelo.conf# traceroute 192.168.0.218
traceroute to 192.168.0.218 (192.168.0.218), 30 hops max, 60 byte packets
connect: Network is unreachable
root@Castelo:/tmp/pycore.38147/Castelo.conf# █

```

Figura 45: Verificação de conexão com os 3 diferentes polos

2. Executamos vários comandos `tracert` nos 3 diferentes polos de forma a verificar que a ligação foi quebrada.

```

root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 192.168.0.192 netmask 255.255.255.248 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 192.168.0.200 netmask 255.255.255.248 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 192.168.0.208 netmask 255.255.255.248 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 192.168.0.216 netmask 255.255.255.248 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# ping 192.168.0.202
PING 192.168.0.202 (192.168.0.202) 56(84) bytes of data.
64 bytes from 192.168.0.202: icmp_seq=1 ttl=55 time=0.372 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.141 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.150 ms
64 bytes from 192.168.0.202: icmp_seq=4 ttl=55 time=0.125 ms
64 bytes from 192.168.0.202: icmp_seq=5 ttl=55 time=0.138 ms
^C
--- 192.168.0.202 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4077ms
rtt min/avg/max/mdev = 0.125/0.185/0.372/0.093 ms
root@Castelo:/tmp/pycore.46603/Castelo.conf# ping 192.168.0.194
PING 192.168.0.194 (192.168.0.194) 56(84) bytes of data.
64 bytes from 192.168.0.194: icmp_seq=1 ttl=55 time=0.205 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=55 time=0.109 ms
64 bytes from 192.168.0.194: icmp_seq=3 ttl=55 time=0.147 ms
64 bytes from 192.168.0.194: icmp_seq=4 ttl=55 time=0.126 ms
^C
--- 192.168.0.194 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.109/0.146/0.205/0.036 ms
root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 172.16.143.0 netmask 255.255.255.248 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 192.168.0.232 netmask 255.255.255.252 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 192.168.0.240 netmask 255.255.255.252 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# route add -net 192.168.0.248 netmask 255.255.255.252 gw 192.168.0.225
root@Castelo:/tmp/pycore.46603/Castelo.conf# ping 192.168.0.234
PING 192.168.0.234 (192.168.0.234) 56(84) bytes of data.
64 bytes from 192.168.0.234: icmp_seq=1 ttl=61 time=0.142 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=61 time=0.067 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=61 time=0.066 ms
64 bytes from 192.168.0.234: icmp_seq=4 ttl=61 time=0.069 ms
^C
--- 192.168.0.234 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.066/0.086/0.142/0.032 ms

```

Figura 46: Adição de rotas e verificação de conexão entre polos

3. Adicionámos as rotas necessárias para estabelecer as ligações entre os diferentes polos e verificamos isso com recurso do comando `ping`, onde as adições de rotas foram bem executadas porque foram estabelecidas as ligações necessárias.

```

root@Castelo:/tmp/pycore.46603/Castelo.conf# netstat -nr
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt  Iface
172.16.143.0     192.168.0.225   255.255.255.248 UG        0 0          0 eth0
192.168.0.192    192.168.0.225   255.255.255.248 UG        0 0          0 eth0
192.168.0.200    192.168.0.225   255.255.255.248 UG        0 0          0 eth0
192.168.0.208    192.168.0.225   255.255.255.248 UG        0 0          0 eth0
192.168.0.216    192.168.0.225   255.255.255.248 UG        0 0          0 eth0
192.168.0.224    0.0.0.0         255.255.255.248 U        0 0          0 eth0
192.168.0.232    192.168.0.225   255.255.255.252 UG        0 0          0 eth0
192.168.0.240    192.168.0.225   255.255.255.252 UG        0 0          0 eth0
192.168.0.248    192.168.0.225   255.255.255.252 UG        0 0          0 eth0
root@Castelo:/tmp/pycore.46603/Castelo.conf# 

```

Figura 47: Tabela de endereçamento


```

root@Castelo:/tmp/pycore.33723/Castelo.conf# netstat -nr
Kernel IP routing table
Destination        Gateway           Genmask          Flags        MSS  Window  irtt  Iface
10.0.0.0            192.168.0.225    255.255.255.252 UG           0 0           0 eth0
10.0.0.4            192.168.0.225    255.255.255.252 UG           0 0           0 eth0
10.0.0.8            192.168.0.225    255.255.255.252 UG           0 0           0 eth0
10.0.0.12           192.168.0.225    255.255.255.252 UG           0 0           0 eth0
10.0.0.16           192.168.0.225    255.255.255.252 UG           0 0           0 eth0
10.0.0.20           192.168.0.225    255.255.255.252 UG           0 0           0 eth0
10.0.0.24           192.168.0.225    255.255.255.252 UG           0 0           0 eth0
10.0.0.28           192.168.0.225    255.255.255.252 UG           0 0           0 eth0
172.16.142.0        192.168.0.225    255.255.255.252 UG           0 0           0 eth0
172.16.142.4        192.168.0.225    255.255.255.252 UG           0 0           0 eth0
172.16.143.0        192.168.0.225    255.255.255.248 UG           0 0           0 eth0
192.168.0.192       192.168.0.225    255.255.255.248 UG           0 0           0 eth0
192.168.0.200       192.168.0.225    255.255.255.248 UG           0 0           0 eth0
192.168.0.208       192.168.0.225    255.255.255.248 UG           0 0           0 eth0
192.168.0.216       192.168.0.225    255.255.255.248 UG           0 0           0 eth0
192.168.0.224       0.0.0.0           255.255.255.248 U           0 0           0 eth0
192.168.0.232       192.168.0.225    255.255.255.252 UG           0 0           0 eth0
192.168.0.240       192.168.0.225    255.255.255.252 UG           0 0           0 eth0
192.168.0.248       192.168.0.225    255.255.255.252 UG           0 0           0 eth0

```

Figura 48: Tabela de endereçamento (atualizada)

4. Na figura 47, podemos visualizar a nossa 1ª tabela de endereçamento. As rotas desta tabela foram insuficientes, para uma boa ligação com os polos "Galiza" e "CDN", visto que faltavam rotas entre "Castelo" e os 2 polos. A falta de conexão foi verificada, logo no router "n5" porque não existia uma rota que ligasse "Castelo" a este, impedindo a ligação com o "Core" e como consequência existia uma falha de ligação com os polos. Por outras palavras, o nó Castelo não conhecia o Core da Rede, pelo que foi necessário adicionar as diversas entradas na sua tabela de encaminhamento para que tivesse conectividade com Galiza e CDN.
5. Por fim, aplicamos o comando `netstat -nr` para obtermos a tabela de encaminhamento de todas as rotas de "Castelo", necessárias para a conexão entre "Castelo" e os 3 diferentes polos (figura 48).

5.2 Por modo a garantir uma posição estrategicamente mais vantajosa e ter casa de férias para relaxar entre batalhas, ordena também a construção de um segundo Castelo, em Braga. Não tendo qualquer queixa do serviço prestado, recorre novamente aos serviços do ISP ReiDaNet para ter acesso à rede no segundo Castelo. O ISP atribuiu-lhe o endereço de rede IP 172.16.XX.128/26 em que XX corresponde ao seu número de grupo (PLXX). Defina um esquema de endereçamento que permita o estabelecimento de pelo menos 3 redes e que garanta que cada uma destas possa ter 10 ou mais hosts. Assuma que todos os endereços de sub-redes são utilizáveis.

O endereço IP do Castelo é, de acordo com o nosso número de grupo, 172.16.107.128/26. Tendo em conta que o endereço IP tem 32 bits e que a máscara de rede é /26, então apenas os últimos 6 bits (da esquerda para a direita) poderão ser manipulados. Como o número subredes deve ser maior ou igual a 3, então devem ser reservados dois bits para representar a subrede. Por consequência, serão necessários 4 bits para representar os hosts. A máscara de rede para subnetting passa então a ser /28 (26+2 bits do ID da subrede).

O esquema de endereçamento passa então a ser o seguinte: nos último octeto (da esquerda para a direita), os primeiros 2 bits são imutáveis, os próximos dois representarão a subrede e os últimos 4 representarão o host. Na tabela abaixo está representada a gama dos endereços de IP presentes nas subredes.

Subrede	Host	Gama de endereços IP
00	0001 a 1110	172.16.107.129 até 172.16.107.142
01	0001 a 1110	172.16.107.145 até 172.16.107.158
10	0001 a 1110	172.16.107.161 até 172.16.107.174

Tabela 1: Esquema de subnetting a partir do endereço IP 172.16.107.1

5.3 Ligue um novo host diretamente ao router ReiDaNet. Associe-lhe um endereço, à sua escolha, pertencente a uma sub-rede disponível das criadas na alínea anterior (garanta que a interface do router ReiDaNet utiliza o primeiro endereço da sub-rede escolhida). Verifique que tem conectividade com os diferentes polos. Existe algum host com o qual não seja possível comunicar? Porquê? Se reiniciou a simulação, repita todas as alterações efetuadas anteriormente e responda a esta questão a partir desse estado.

De acordo com a figura abaixo, podemos observar o host **HostSubnet** que está ligado diretamente ao ISP **ReiDaNet**, cujos endereços estão consoante a nossa subrede #1 do nosso esquema de endereçamento. Ou seja, a interface do ReiDaNet contém o primeiro endereço da gama destinada a esta subrede e o **HostSubnet** tem o segundo endereço. Também é possível observar que este host tem conectividade a todos os hosts. Isto justifica-se pelo facto de todos os routers (tanto ISP's como routers de acesso como routers do core da rede) conhecerem todos os endereços de todas as redes, pelo que a criação de uma nova subrede num qualquer router irá permitir a conexão desse local para qualquer outro local da topologia da rede.

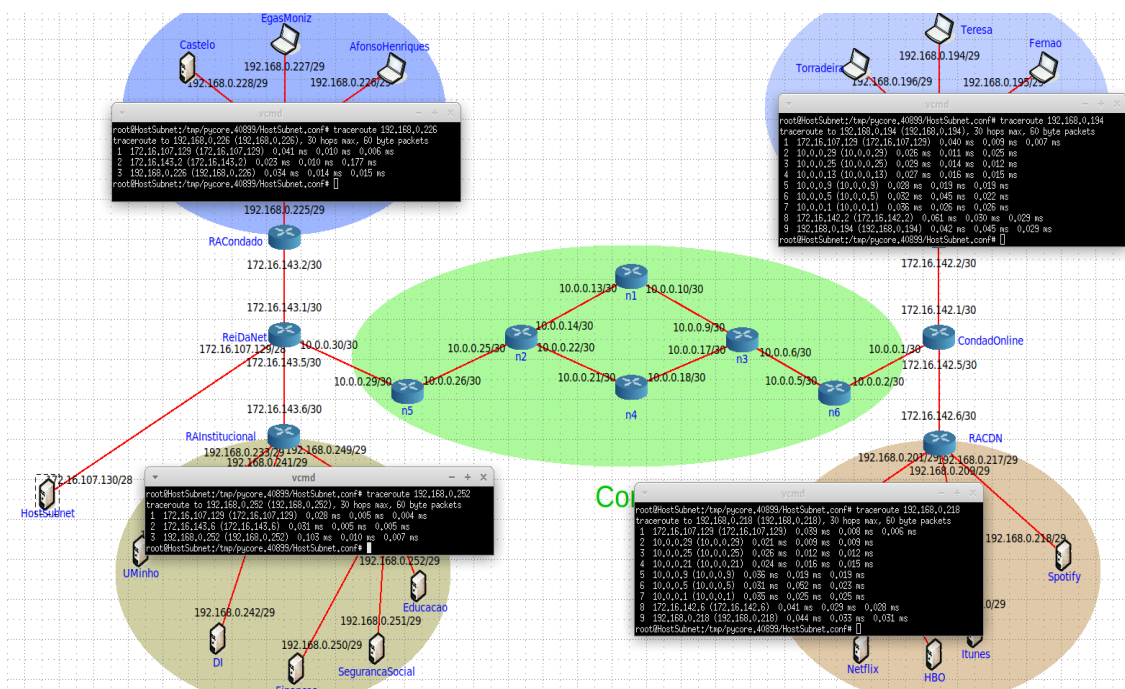


Figura 49: Ligação de um novo host e traceroute para todos os polos

6

6.1 De modo a facilitar a travessia, elimine as rotas referentes a Galiza e CDN no dispositivo n6 e defina um esquema de sumarização de rotas (Supernetting) que permita o uso de apenas uma rota para ambos os polos. Confirme que a conectividade é mantida.

Para fazer supernetting dos endereços IP dados, precisamos primeiro identificar qual é a máscara de sub-rede apropriada para agregar as subredes numa única rota.

Podemos converter cada endereço IP para a sua forma binária e começamos por identificar os bits comuns em cada endereço. Os endereços IP dados são:

- ★ 192.168.0.192/29 (binário: 11000000 10101000 00000000 11000000)
- ★ 192.168.0.200/29 (binário: 11000000 10101000 00000000 11001000)
- ★ 192.168.0.208/29 (binário: 11000000 10101000 00000000 11010000)
- ★ 192.168.0.216/29 (binário: 11000000 10101000 00000000 11011000)

Nota: /29, corresponde ao netmask 255.255.255.248.

Ao observar estes endereços IP em binário, podemos verificar que os primeiros 27 bits são os mesmos em todos eles. Isto significa que podemos usar uma máscara de subrede /27 para os agregar.

A máscara /27 significa que os primeiros 27 bits do endereço IP são o prefixo de rede, enquanto os últimos 5 bits são o sufixo do host. Isto dá um total de 2^5 (32) endereços de host disponíveis em cada subrede.

Portanto, podemos resumir as quatro sub-redes numa única rota usando o seguinte endereço:

- ★ 192.168.0.192/27

Nota: /27, corresponde ao netmask 255.255.255.224.

Isto permite aos dispositivos nessas subredes que usem uma única rota comunicar entre si.

Usando o comando **netstat -nr**, obtivemos as tabelas de encaminhamento do dispositivo n6.

```
root@n6:/tmp/pycore.38825/n6.conf# netstat -nr
```

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
10.0.0.0	0.0.0.0	255.255.255.252	U	0	0	0	eth0
10.0.0.4	0.0.0.0	255.255.255.252	U	0	0	0	eth1
10.0.0.8	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
10.0.0.12	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
10.0.0.16	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
10.0.0.20	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
10.0.0.24	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
10.0.0.28	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
172.0.0.0	10.0.0.6	255.0.0.0	UG	0	0	0	eth1
172.16.142.0	10.0.0.1	255.255.255.252	UG	0	0	0	eth0
172.16.142.4	10.0.0.1	255.255.255.252	UG	0	0	0	eth0
172.16.143.0	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
172.16.143.4	10.0.0.6	255.255.255.252	UG	0	0	0	eth1
192.168.0.192	10.0.0.1	255.255.255.224	UG	0	0	0	eth0
192.168.0.224	10.0.0.6	255.255.255.248	UG	0	0	0	eth1
192.168.0.232	10.0.0.6	255.255.255.248	UG	0	0	0	eth1
192.168.0.240	10.0.0.6	255.255.255.248	UG	0	0	0	eth1
192.168.0.248	10.0.0.6	255.255.255.248	UG	0	0	0	eth1

Figura 50: Tabela de endereçamento resultante no n6

Para verificar se a conexão foi bem estabelecida, nas figuras em baixo, temos o traceroute e o ping para 1 ponto de cada um dos polos envolvidos neste exercício.

```

root@n6:/tmp/pycore.38825/n6.conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.976 ms 0.009 ms 0.003 ms
 2 172.16.142.2 (172.16.142.2) 0.160 ms 0.010 ms 0.007 ms
 3 192.168.0.194 (192.168.0.194) 0.242 ms 0.010 ms 0.006 ms
root@n6:/tmp/pycore.38825/n6.conf# ping 192.168.0.194
PING 192.168.0.194 (192.168.0.194) 56(84) bytes of data:
64 bytes from 192.168.0.194: icmp_seq=1 ttl=62 time=0.135 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=62 time=0.077 ms
64 bytes from 192.168.0.194: icmp_seq=3 ttl=62 time=0.080 ms
64 bytes from 192.168.0.194: icmp_seq=4 ttl=62 time=0.120 ms
^C
--- 192.168.0.194 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3062ms
rtt min/avg/max/mdev = 0.077/0.103/0.135/0.025 ms

```

Figura 51: Traceroute e ping na "Teresa", no polo "Galiza"

```

root@n6:/tmp/pycore.38825/n6.conf# traceroute 192.168.0.218
traceroute to 192.168.0.218 (192.168.0.218), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.092 ms 0.080 ms 0.008 ms
 2 172.16.142.6 (172.16.142.6) 0.405 ms 0.024 ms 0.005 ms
 3 192.168.0.218 (192.168.0.218) 0.673 ms 0.036 ms 0.006 ms
root@n6:/tmp/pycore.38825/n6.conf# ping 192.168.0.218
PING 192.168.0.218 (192.168.0.218) 56(84) bytes of data:
64 bytes from 192.168.0.218: icmp_seq=1 ttl=62 time=0.233 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=62 time=2.07 ms
64 bytes from 192.168.0.218: icmp_seq=3 ttl=62 time=0.077 ms
64 bytes from 192.168.0.218: icmp_seq=4 ttl=62 time=0.108 ms
^C
--- 192.168.0.218 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3038ms
rtt min/avg/max/mdev = 0.077/0.622/2.073/0.839 ms

```

Figura 52: Traceroute e ping na "Spotify", no polo "CDN"

6.2 Repita o processo descrito na alínea anterior para CondadoPortucalense e Institucional, também no dispositivo n6.

Para fazer supernetting dos endereços IP dados, precisamos primeiro identificar qual é a máscara de subrede apropriada para agregar as subredes numa única rota.

Podemos converter cada endereço IP para a sua forma binária e começamos por identificar os bits comuns em cada endereço. Os endereços IP dados são:

- ★ 192.168.0.224/29 (binário: 11000000 10101000 00000000 11100000)
- ★ 192.168.0.232/29 (binário: 11000000 10101000 00000000 11101000)
- ★ 192.168.0.240/29 (binário: 11000000 10101000 00000000 11110000)
- ★ 192.168.0.248/29 (binário: 11000000 10101000 00000000 11111000)

Nota: /29, corresponde ao netmask 255.255.255.248.

Ao observar estes endereços IP em binário, podemos verificar que os primeiros 27 bits são os mesmos em todos eles. Isto significa que podemos usar uma máscara de subrede /27 para os agregar.

A máscara /27 significa que os primeiros 27 bits do endereço IP são o prefixo de rede, enquanto os últimos 5 bits são o sufixo do host. Isto dá um total de 2^5 (32) endereços de host disponíveis em cada subrede.

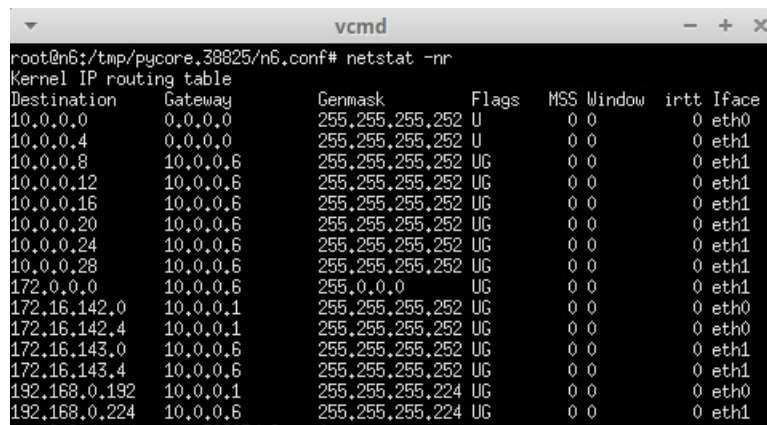
Portanto, podemos resumir as quatro subredes numa única rota usando o seguinte endereço:

- ★ 192.168.0.224/27

Nota: /27, corresponde ao netmask 255.255.255.224.

Isto permite aos dispositivos nessas subredes que usem uma única rota comunicar entre si.

Usando o comando **netstat -nr**, obtivemos as tabelas de encaminhamento do dispositivo n6.



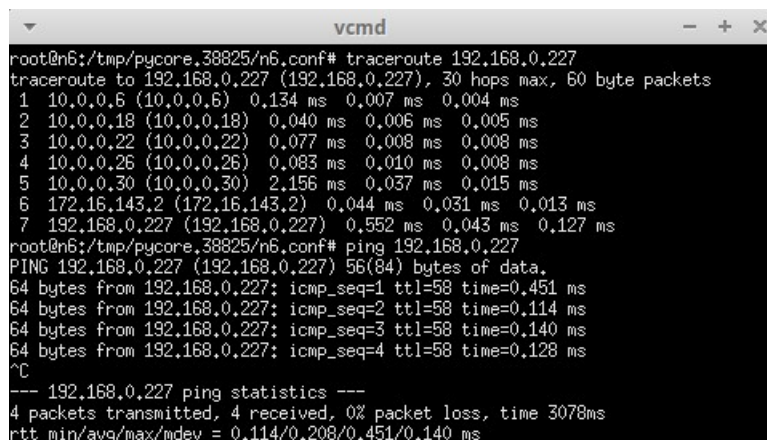
```

root@n6:/tmp/pycore.38825/n6.conf# netstat -nr
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0          0.0.0.0         255.255.255.252 U        0 0        0 eth0
10.0.0.4          0.0.0.0         255.255.255.252 U        0 0        0 eth1
10.0.0.8          10.0.0.6        255.255.255.252 UG       0 0        0 eth1
10.0.0.12         10.0.0.6        255.255.255.252 UG       0 0        0 eth1
10.0.0.16         10.0.0.6        255.255.255.252 UG       0 0        0 eth1
10.0.0.20         10.0.0.6        255.255.255.252 UG       0 0        0 eth1
10.0.0.24         10.0.0.6        255.255.255.252 UG       0 0        0 eth1
10.0.0.28         10.0.0.6        255.255.255.252 UG       0 0        0 eth1
172.0.0.0         10.0.0.6        255.0.0.0       UG       0 0        0 eth1
172.16.142.0      10.0.0.1        255.255.255.252 UG       0 0        0 eth0
172.16.142.4      10.0.0.1        255.255.255.252 UG       0 0        0 eth0
172.16.143.0      10.0.0.6        255.255.255.252 UG       0 0        0 eth1
172.16.143.4      10.0.0.6        255.255.255.252 UG       0 0        0 eth1
192.168.0.192     10.0.0.1        255.255.255.224 UG       0 0        0 eth0
192.168.0.224     10.0.0.6        255.255.255.224 UG       0 0        0 eth1

```

Figura 53: Tabela de endereçamento resultante no n6

Para verificar se a conexão foi bem estabelecida, nas figuras em baixo, temos o traceroute e o ping para 1 ponto de cada um dos polos envolvidos neste exercício.



```

root@n6:/tmp/pycore.38825/n6.conf# traceroute 192.168.0.227
traceroute to 192.168.0.227 (192.168.0.227), 30 hops max, 60 byte packets
 1 10.0.0.6 (10.0.0.6) 0.134 ms 0.007 ms 0.004 ms
 2 10.0.0.18 (10.0.0.18) 0.040 ms 0.006 ms 0.005 ms
 3 10.0.0.22 (10.0.0.22) 0.077 ms 0.008 ms 0.008 ms
 4 10.0.0.26 (10.0.0.26) 0.083 ms 0.010 ms 0.008 ms
 5 10.0.0.30 (10.0.0.30) 2.156 ms 0.037 ms 0.015 ms
 6 172.16.143.2 (172.16.143.2) 0.044 ms 0.031 ms 0.013 ms
 7 192.168.0.227 (192.168.0.227) 0.552 ms 0.043 ms 0.127 ms
root@n6:/tmp/pycore.38825/n6.conf# ping 192.168.0.227
PING 192.168.0.227 (192.168.0.227) 56(84) bytes of data.
64 bytes from 192.168.0.227: icmp_seq=1 ttl=58 time=0.451 ms
64 bytes from 192.168.0.227: icmp_seq=2 ttl=58 time=0.114 ms
64 bytes from 192.168.0.227: icmp_seq=3 ttl=58 time=0.140 ms
64 bytes from 192.168.0.227: icmp_seq=4 ttl=58 time=0.128 ms
^C
--- 192.168.0.227 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.114/0.208/0.451/0.140 ms

```

Figura 54: Traceroute e ping na "EgasMoniz", no polo "Condado Portucalense"

```

root@n6:/tmp/pycore.38825/n6.conf# traceroute 192.168.0.234
traceroute to 192.168.0.234 (192.168.0.234), 30 hops max, 60 byte packets
 1 10.0.0.6 (10.0.0.6) 0.147 ms 0.005 ms 0.161 ms
 2 10.0.0.10 (10.0.0.10) 0.140 ms 0.008 ms 0.006 ms
 3 10.0.0.22 (10.0.0.22) 0.034 ms 0.007 ms 0.007 ms
 4 10.0.0.26 (10.0.0.26) 0.017 ms 0.009 ms 0.009 ms
 5 10.0.0.30 (10.0.0.30) 0.016 ms 0.010 ms 0.010 ms
 6 172.16.143.6 (172.16.143.6) 0.088 ms 0.067 ms 0.014 ms
 7 192.168.0.234 (192.168.0.234) 0.395 ms 0.032 ms 0.015 ms
root@n6:/tmp/pycore.38825/n6.conf# ping 192.168.0.234
PING 192.168.0.234 (192.168.0.234) 56(84) bytes of data.
64 bytes from 192.168.0.234: icmp_seq=1 ttl=58 time=0.077 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=58 time=0.120 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=58 time=0.245 ms
64 bytes from 192.168.0.234: icmp_seq=4 ttl=58 time=0.123 ms
^C
--- 192.168.0.234 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3063ms
rtt min/avg/max/mdev = 0.077/0.141/0.245/0.062 ms

```

Figura 55: Traceroute e ping na "Uminho", no polo "Institucional"

6.3 Comente os aspetos positivos e negativos do uso do Supernetting.

O supernetting é uma técnica que apresenta diversos benefícios para as redes de computadores, tais como a redução do tamanho das tabelas de routing, o aumento da eficiência de routing por causa da aceleração da tomada de decisões, a diminuição do tamanho e da frequência das tabelas de routing, a libertação de memória e a redução da potência de processamento, além de reduzir o tráfego na rede e auxiliar na gestão mais eficaz da rede.

No entanto, existem também alguns aspectos negativos associados ao supernetting, como a necessidade de que as redes da supernet utilizem a mesma classe de endereçamento IP, a possibilidade de um problema num único endereço de IP afetar vários hosts e a necessidade de configurar manualmente rotas para cada subrede.