



Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Redes de Computadores

Ano Letivo de 2023/2024

Trabalho Prático Nº 2 Protocolo IPv4 Grupo 74 Datagramas IP e Fragmenta- ção

Tomás Henrique Alves Melo (A104529)
José Pedro Torres Vasconcelos (A100763)
Sandro José Rodrigues Coelho (A105672)

9 de abril de 2024

RC

Índice

1	Parte 1	1
1.1	Questão 1	1
1.1.1	a	1
1.1.2	b	2
1.1.3	c	3
1.1.4	d	4
1.2	Questão 2	4
1.2.1	a	5
1.2.2	b	5
1.2.3	c	6
1.2.4	d	6
1.2.5	e	7
1.2.6	f	7
1.2.7	g	8
1.2.8	h	9
1.3	Questão 3	9
1.3.1	a	10
1.3.2	b	10
1.3.3	c	11
1.3.4	d	11
1.3.5	e	11
1.3.6	f	13
1.3.7	g	13
1.3.8	h	14
1.3.9	i	14
1.3.10	j	14
2	Parte 2	16
2.1	Questão 1	16
2.1.1	a	16
2.1.2	b	17
2.1.3	c	18
2.2	Questão 2	18
2.2.1	a	19
2.2.2	b	21
2.2.3	c	22

2.2.4	d	24
2.2.5	e	26
2.2.6	f	27
2.2.7	g	28
2.3	Questão 3	28
2.3.1	a	28
2.3.2	b	29
2.3.3	c	30
3	Conclusão	31

Lista de Figuras

1.1	Topologia CORE criada	1
1.2	<i>Traceroute</i> na <i>shell</i> de Jasmine pelo comando <i>traceroute</i>	2
1.3	Imagem vista pelo Wireshark dos pacotes	2
1.4	Rota mínima observada	3
1.5	Comando <i>traceroute</i> com opção <i>-q</i> para cálculo do valor médio do tempo de ida-e-volta	4
1.6	Imagem capturada do Wireshark assinalando o endereço IP pedido	5
1.7	Valor do campo <i>protocol</i>	5
1.8	Header Length e Total Length	6
1.9	Flags de fragmentação	6
1.10	Pacotes Ordenados	7
1.11	Pacotes Ordenados por 'endereço destino'	8
1.12	Resultados do comando <i>ping -s 3740 marco.uminho.pt</i>	9
1.13	Primeira mensagem ICMP	10
1.14	Informação relativa ao primeiro segmento	10
1.15	Informação relativa ao segundo segmento	11
1.16	Representação gráfica da fragmentação feita com indicação dos respetivos 'Fragment Offsets'	12
1.17	Informação relativa ao terceiro segmento	13
1.18	Ping efetuado que mostra o valor máximo sem fragmentação	15
2.1	Representação da ligação de Castelo2 diretamente ao router ReiDaNet	17
2.2	Análise da conectividade com os dispositivos do Condado Portucalense	17
2.3	Tabela de encaminhamento	18
2.4	Conectividade com o servidor 'Finanças'	19
2.5	Conectividade com o servidor 'HBO'	19
2.6	Conectividade com o servidor 'iTunes'	20
2.7	Conectividade com o servidor 'Netflix'	20
2.8	Conectividade com o servidor 'Spotify'	20
2.9	Conectividade com o servidor 'Youtube'	21
2.10	Tabela de encaminhamento do dispositivo AfonsoHenriques	22
2.11	Tabela de encaminhamento do dispositivo Teresa	22
2.12	Comando <i>add</i> no n5	22
2.13	Comando <i>del</i> no n2	22
2.14	Comandos <i>del</i> e <i>add</i> no n1	23
2.15	Comando <i>traceroute</i> no AfonsoHenriques 1º passo	23

2.16	Comando <i>traceroute</i> no AfonsoHenriques 4º passo	23
2.17	Add do route de regresso em 'RAGaliza'	24
2.18	Netstat de RAGaliza	24
2.19	Imagem retirada do Wireshark que mostra os pacotes	24
2.20	<i>Traceroute</i> dos caminhos usando o comando <i>traceroute</i>	25
2.21	Representação do caminho e dos vários sentidos	26
2.22	Entrada referido no enunciado	26
2.23	<i>Traceroute</i> e <i>netstat</i> no sentido n3-Galiza	27
2.24	Conectividade após <i>Supernetting</i>	29
2.25	Conectividade após <i>Supernetting</i>	30

1 Parte 1

1.1 Questão 1

"Prepare uma topologia CORE para verificar o comportamento do traceroute. Na topologia deve existir: um host (pc) cliente designado Jasmine, cujo router de acesso é RA1; o router RA1 está simultaneamente ligado a dois routers no core da rede RCx e RCy (cada grupo de trabalho deve personalizar a rede de core fazendo x e y corresponder ao seu identificador de grupo PLxy); estes estão conectados a um router de acesso RA2, que por sua vez, se liga a um host (servidor) designado Aladdin. Ajuste o nome dos equipamentos atribuídos por defeito para o enunciado. Apenas nas ligações (links) da rede de core, estabeleça um tempo de propagação de 20 ms. Após ativar a topologia, note que pode não existir conectividade IP imediata entre Jasmine e Aladdin pois é necessário que o anúncio de rotas entre routers se efetue e estabilize."

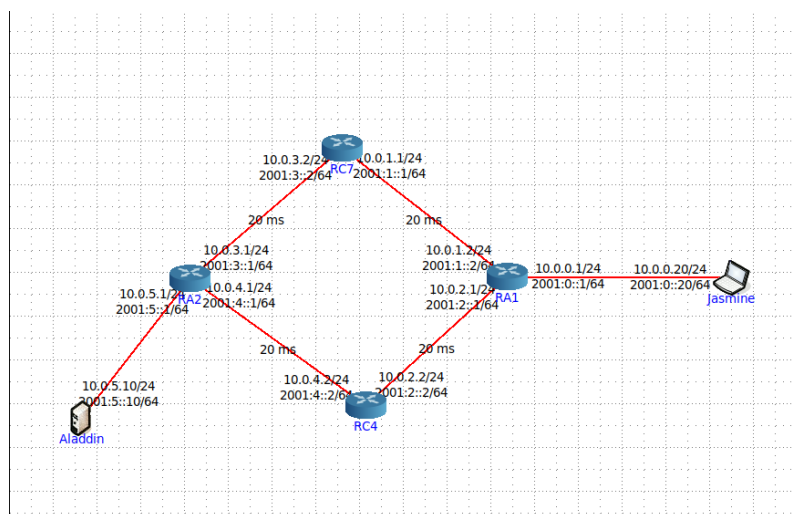


Figura 1.1: Topologia CORE criada

1.1.1 a

"Active o Wireshark no host Jasmine. Numa shell de Jasmine execute o comando `traceroute -I` para o endereço IP do Aladdin. Registe e analise o tráfego ICMP enviado pelo sistema Jasmine

e o tráfego ICMP recebido como resposta. Explique os resultados obtidos tendo em conta o princípio de funcionamento do traceroute.”

```
vcmd
root@Jasmine:/tmp/pycore.35751/Jasmine.conf# traceroute -I 10.0.0.10
traceroute to 10.0.0.10 (10.0.0.10), 30 hops max, 60 byte packets
 1 10.0.5.1 (10.0.5.1) 0.093 ms 0.004 ms 0.002 ms
 2 10.0.2.2 (10.0.2.2) 40.300 ms 40.294 ms 40.294 ms
 3 10.0.1.2 (10.0.1.2) 80.831 ms 80.831 ms 80.830 ms
 4 10.0.0.10 (10.0.0.10) 80.828 ms 80.828 ms 80.826 ms
root@Jasmine:/tmp/pycore.35751/Jasmine.conf#
```

Figura 1.2: Traceroute na shell de Jasmine pelo comando traceroute

71	65.090587387	10.0.5.1	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
72	65.090592476	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=2/512, ttl=1 (no response found!)
73	65.090594295	10.0.5.1	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
74	65.090595014	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=3/768, ttl=1 (no response found!)
75	65.090597080	10.0.5.1	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
76	65.090598413	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=4/1024, ttl=2 (no response found!)
77	65.090604030	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=5/1280, ttl=2 (no response found!)
78	65.090605624	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=6/1536, ttl=2 (no response found!)
79	65.090606964	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=7/1792, ttl=3 (no response found!)
80	65.090608215	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=8/2048, ttl=3 (no response found!)
81	65.090609686	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=9/2304, ttl=3 (no response found!)
82	65.090610948	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=10/2560, ttl=4 (reply in 101)
83	65.090612050	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=11/2816, ttl=4 (reply in 102)
84	65.090613982	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=12/3072, ttl=4 (reply in 103)
85	65.090615190	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=13/3328, ttl=5 (reply in 104)
86	65.090617344	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=14/3584, ttl=5 (reply in 105)
87	65.090618374	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=15/3840, ttl=5 (reply in 106)
88	65.090619525	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=16/4096, ttl=6 (reply in 107)
89	65.090620878	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=17/4352, ttl=6 (reply in 108)
90	65.090631895	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=18/4608, ttl=6 (reply in 109)
91	65.090633436	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=19/4864, ttl=7 (reply in 110)
92	65.130890280	10.0.2.2	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
93	65.130897856	10.0.2.2	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
94	65.130898175	10.0.2.2	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
95	65.131050841	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=20/5120, ttl=7 (reply in 111)
96	65.131055348	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=21/5376, ttl=7 (reply in 112)
97	65.131056067	10.0.5.20	10.0.0.10	ICMP	74 Echo (ping) request id=0x001c, seq=22/5632, ttl=8 (reply in 113)
98	65.171435683	10.0.1.2	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
99	65.171438395	10.0.1.2	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
100	65.171438675	10.0.1.2	10.0.5.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
101	65.171438949	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=10/2560, ttl=61 (request in 82)
102	65.171439214	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=11/2816, ttl=61 (request in 83)
103	65.171439444	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=12/3072, ttl=61 (request in 84)
104	65.171439776	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=13/3328, ttl=61 (request in 85)
105	65.171439997	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=14/3584, ttl=61 (request in 86)
106	65.171440215	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=15/3840, ttl=61 (request in 87)
107	65.171440431	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=16/4096, ttl=61 (request in 88)
108	65.171440602	10.0.0.10	10.0.5.20	ICMP	74 Echo (ping) reply id=0x001c, seq=17/4352, ttl=61 (request in 89)

Figura 1.3: Imagem vista pelo Wireshark dos pacotes

R: O *traceroute* serve-se da alteração do *TTL* dos pacotes *IP* em valor crescente, entregando em cada *router* uma mensagem *ICMP* quando o *TTL* é nulo ou seja, quando há o descarte do pacote. A partir do *host* em que o *traceroute* foi realizado, as mensagens *ICMP* têm a capacidade de instituir a rota de acesso ao *Alladin*. O tráfego *ICMP* enviado do sistema *Jasmine* para o sistema *Alladin* (10.0.0.10) diz respeito ao envio de 3 datagramas de cada vez com o mesmo *TTL*, que aumenta em 1, após cada três datagramas enviados. O tráfego recebido corresponde a um datagrama de resposta por cada pedido, sempre com *TTL* = 61.

1.1.2 b

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

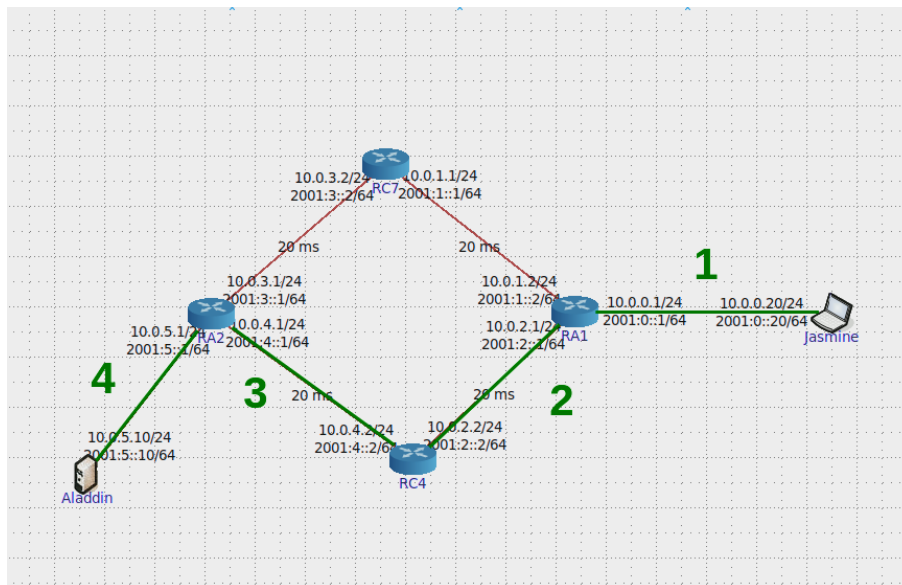
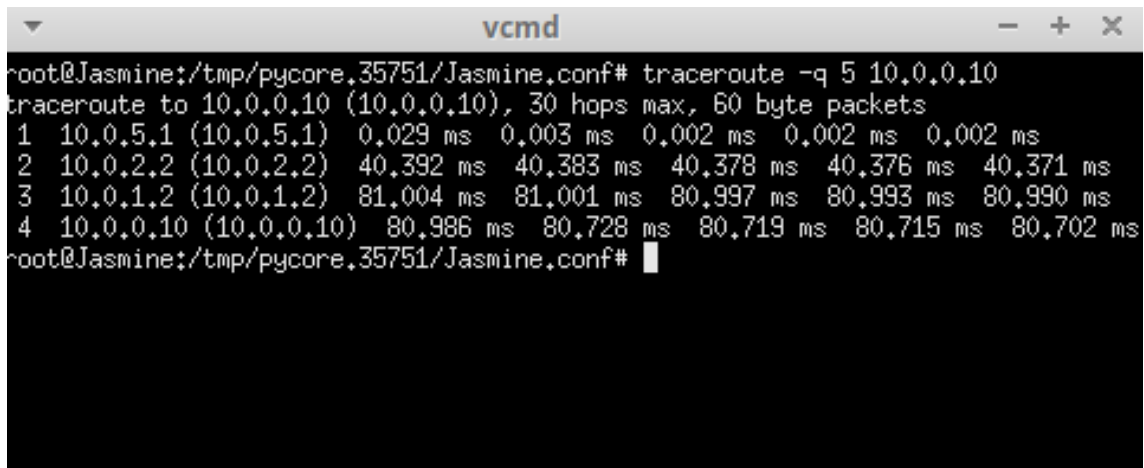


Figura 1.4: Rota mínima observada

R: O valor inicial mínimo do campo *TTL* é necessariamente 4. Através do esquema acima é possível perceber que um datagrama que parta do *host Jasmine* necessita de 4 saltos até chegar ao servidor *Alladin*.

1.1.3 c

"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*."



```
root@Jasmine:/tmp/pycore.35751/Jasmine.conf# traceroute -q 5 10.0.0.10
traceroute to 10.0.0.10 (10.0.0.10), 30 hops max, 60 byte packets
 1  10.0.5.1 (10.0.5.1)  0.029 ms  0.003 ms  0.002 ms  0.002 ms  0.002 ms
 2  10.0.2.2 (10.0.2.2)  40.392 ms  40.383 ms  40.378 ms  40.376 ms  40.371 ms
 3  10.0.1.2 (10.0.1.2)  81.004 ms  81.001 ms  80.997 ms  80.993 ms  80.990 ms
 4  10.0.0.10 (10.0.0.10)  80.986 ms  80.728 ms  80.719 ms  80.715 ms  80.702 ms
root@Jasmine:/tmp/pycore.35751/Jasmine.conf#
```

Figura 1.5: Comando traceroute com opção -q para cálculo do valor médio do tempo de ida-e-volta

R: O valor da média do *RTT* equivale a 80.770 milissegundos.

1.1.4 d

"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?"

R: Para calcular o *One-Way Delay* (*OWD*), que representa o tempo de chegada de um pacote ao destino numa só direção, é preciso considerar variáveis como latência de rede, congestionamento e também os diferentes meios de transporte. Ora, a simples divisão do *RTT* por dois não é suficiente, devido à complexidade desses fatores em cenários reais.

1.2 Questão 2

Pretende-se agora usar o *traceroute* na sua máquina nativa e gerar datagramas *IP* de diferentes tamanhos. **Windows:** O programa *tracert* disponibilizado no *Windows* não permite mudar o tamanho das mensagens a enviar. Como alternativa, o programa *pingplotter* (ou equivalente) na sua versão livre ou *shareware* (<http://www.pingplotter.com>) permite maior flexibilidade para efetuar *traceroute*. Descarregue, instale e experimente o *pingplotter* face ao objetivo pretendido. O tamanho da mensagem a enviar (*ICMP Echo Request*) pode ser estabelecido no *pingplotter* no menu Edit -> Options -> Default Settings -> Engine. Uma vez enviado um conjunto de pacotes com valores crescentes de *TTL*, o programa recomeça com *TTL*=1, após um determinado intervalo. Tanto o valor do intervalo de tempo como o número de intervalos podem ser configurados. **Linux/Unix:** O comando *traceroute* permite indicar o tamanho do

pacote *ICMP* (opção *-I*) através da linha de comando, a seguir ao *host* de destino (ver *man traceroute*). Exemplo: Documente as suas respostas com a impressão do(s) output(s) (e.g. pacote(s)) que as suportam. Procedimento a seguir: Usando o *Wireshark* capture o tráfego gerado pelo *traceroute* sem especificar o tamanho do pacote, i.e., quando é usado o tamanho do pacote de prova por defeito. Utilize como máquina destino o *host marco.uminho.pt*. Pare a captura. Com base no tráfego capturado, identifique os pedidos ***ICMP Echo Request*** e o conjunto de mensagens devolvidas como resposta. Selecione a primeira mensagem *ICMP* capturada e centre a análise no nível protocolar *IP* e, em particular, do cabeçalho *IP* (expanda o *tab* correspondente na janela de detalhe do *Wireshark*). Documente e justifique todas as respostas às seguintes alíneas:

1.2.1 a

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

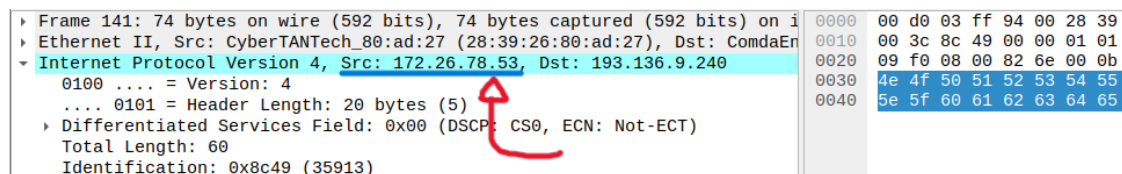


Figura 1.6: Imagem capturada do Wireshark assinalando o endereço *IP* pedido

R: O endereço *IP* da interface ativa do computador utilizado é 172.26.78.53 .

1.2.2 b

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

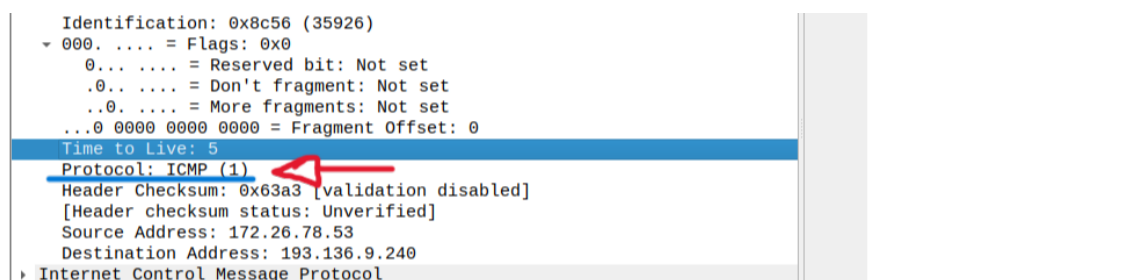


Figura 1.7: Valor do campo *protocol*

R: O valor do campo protocolo é *ICMP* (1). Este valor do campo *protocol* permite estabelecer o protocolo a ser usado. Neste caso, o protocolo, tem como função comunicar informações da camada de rede, permitindo receber relatórios de erros à fonte original.

1.2.3 c

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

<ul style="list-style-type: none"> Internet Protocol Version 4, Src: 172.26.78.53, Dst: 193.136.9.240 <ul style="list-style-type: none"> 0100 = Version: 4 0101 = Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x8c56 (35926) 0000. = Flags: 0x0 <ul style="list-style-type: none"> 0... = Reserved bit: Not set .0... = Don't fragment: Not set ..0. = More fragments: Not set ...0 0000 0000 0000 = Fragment Offset: 0 	0020 09 f0 08 00 82 61 00 00 0030 4e 4f 50 51 52 53 54 55 0040 5e 5f 60 61 62 63 64 65
---	--

Figura 1.8: Header Length e Total Length

R: O cabeçalho *IPv4* tem 20 bytes (*Header Length: 20 bytes*). O campo de dados do datagrama tem 40 bytes. O tamanho do *payload* pode ser calculado pela diferença entre o comprimento total equivalente a 60 bytes (*Total Length: 60 bytes*) e o cabeçalho *IPv4* que equivale a 20 bytes. (60-20 = 40)

1.2.4 d

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

<ul style="list-style-type: none"> Internet Protocol Version 4, Src: 172.26.78.53, Dst: 193.136.9.240 <ul style="list-style-type: none"> 0100 = Version: 4 0101 = Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x8c56 (35926) 0000. = Flags: 0x0 <ul style="list-style-type: none"> 0... = Reserved bit: Not set .0... = Don't fragment: Not set ..0. = More fragments: Not set ...0 0000 0000 0000 = Fragment Offset: 0 	0020 09 f0 08 00 82 61 00 0b 0030 4e 4f 50 51 52 53 54 55 0040 5e 5f 60 61 62 63 64 65
---	--

Figura 1.9: Flags de fragmentação

R: O datagrama *IP* não foi fragmentado. Tal é indicado pelos campos '*More fragments*' e '*Don't fragment*' em que estão ambos definidos como '*Not set*'.

'More fragments = Not set', para além de indicar que este datagrama não é um fragmento intermediário, indica, também, que não há necessidade de mais fragmentos para compor o datagrama original. Porém, o valor do campo 'Fragment Offset' está como Not Set, informando que estamos diante do primeiro pacote. Apesar do campo 'Don't fragment' permitir a fragmentação, não houve tal necessidade neste caso específico. Esta junção de ideias, permite-nos concluir que este é o datagrama original e que não há mais fragmentos a seguir.

1.2.5 e

"Ordene os pacotes capturados de acordo com o endereço IP fonte (e.g., selecionando o cabeçalho da coluna Source), e analise a sequência de tráfego ICMP gerado a partir do endereço IP atribuído à interface da sua máquina. Para a sequência de mensagens ICMP enviadas pelo seu computador, indique que campos do cabeçalho IP variam de pacote para pacote."

R: É possível observar assim que os únicos campos variados são os campos de identificação e de TTL.

No.	Time	Source	Destination	Protocol	Length	Info
158	46.985765796	172.16.2.1	172.26.78.53	ICMP	70	Time-to-live exceeded
161	46.985765880	172.16.2.1	172.26.78.53	ICMP	70	Time-to-live exceeded
186	47.106033741	172.26.254.254	172.26.78.53	ICMP	70	Time-to-live exceeded
187	47.106033818	172.26.254.254	172.26.78.53	ICMP	70	Time-to-live exceeded
188	47.106033850	172.26.254.254	172.26.78.53	ICMP	70	Time-to-live exceeded
141	46.983987650	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
142	46.984012619	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
143	46.984041446	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
144	46.984058644	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
145	46.984073959	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
146	46.984089436	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
147	46.984138523	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
148	46.984152580	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
149	46.984188371	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
150	46.984203619	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
151	46.984215752	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
152	46.984228187	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
153	46.984239946	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
154	46.984251858	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
155	46.984264677	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
156	46.984329844	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
159	46.985765824	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
160	46.985765852	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply

Figura 1.10: Pacotes Ordenados

1.2.6 f

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

R: Verifica-se que a identificação de cada pacote aumenta 1 em valor hexadecimal e no campo TTL o valor é incrementado em 1, a cada 3 mensagens enviadas.

1.2.7 g

"Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP TTL Exceeded enviadas ao seu computador."

No.	Time	Source	Destination	Protocol	Length	Info
157	46.985765310	172.16.2.1	172.26.78.53	ICMP	70	Time-to-live exceeded
158	46.985765796	172.16.2.1	172.26.78.53	ICMP	70	Time-to-live exceeded
159	46.985765824	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
160	46.985765852	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
161	46.985765880	172.16.2.1	172.26.78.53	ICMP	70	Time-to-live exceeded
162	46.985765908	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
163	46.985765935	172.16.115.252	172.26.78.53	ICMP	70	Time-to-live exceeded
164	46.985765964	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
165	46.985881635	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
166	46.985881705	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
167	46.985881741	193.136.9.240	172.26.78.53	ICMP	74	Echo (ping) reply
168	46.985881770	172.16.115.252	172.26.78.53	ICMP	70	Time-to-live exceeded
169	46.986106854	172.16.115.252	172.26.78.53	ICMP	70	Time-to-live exceeded
186	47.106033741	172.26.254.254	172.26.78.53	ICMP	70	Time-to-live exceeded
187	47.106033818	172.26.254.254	172.26.78.53	ICMP	70	Time-to-live exceeded
188	47.106033850	172.26.254.254	172.26.78.53	ICMP	70	Time-to-live exceeded

Figura 1.11: Pacotes Ordenados por 'endereço destino'

i)

"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?"

R: Confirma-se que o valor do campo TTL será 255 nas primeiras três mensagens, decrescendo para 254 nas seguintes três mensagens e decrescendo novamente para 253 nas últimas três mensagens do tipo ICMP TTL exceeded. Assim, é possível concluir que a mesma decremente 1 unidade a cada três mensagens desse tipo. Isto acontece devido à alteração nas condições da própria rede. É possível que um router não seja o mesmo a ser escolhido de seguida para o envio de pacotes. Quando isso acontece, o pacote em questão pode percorrer um caminho que envolve um número diferente de routers, causando um TTL diferente ao anterior.

ii)

"Porque razão as mensagens de resposta ICMP TTL Exceeded são sempre enviadas na origem com um valor TTL relativamente alto?"

R: Considerando que as mensagens ICMP TTL Exceeded têm a finalidade de informar ao host de origem sobre a rejeição de pacotes devido ao TTL excedido, utilizar um valor de TTL alto nessas mensagens assegura que o host será notificado, independentemente do número de saltos na rede. Se o TTL fosse baixo, haveria o risco de a mensagem ser descartada no retorno para

o *host* de origem, causando na não receção da notificação sobre a entrega falhada do pacote.

1.2.8 h

"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?"

R: É possível integrar os dados do cabeçalho *ICMP* diretamente no cabeçalho *IPv4*. Sendo uma das principais vantagens, economizar o processamento do *payload* do datagrama *IP*, pois a informação presente não seria encapsulada no *payload* do datagrama *IP*. No entanto, esta adaptação não é de todo a mais vantajosa para estes casos, dado que pode afetar situações em que o *ICMP* não é necessário, ocupando espaço dentro do cabeçalho *IP*. Concluindo, qualquer alteração no cabeçalho *ICMP* implica uma atualização no protocolo *IPv4*.

1.3 Questão 3

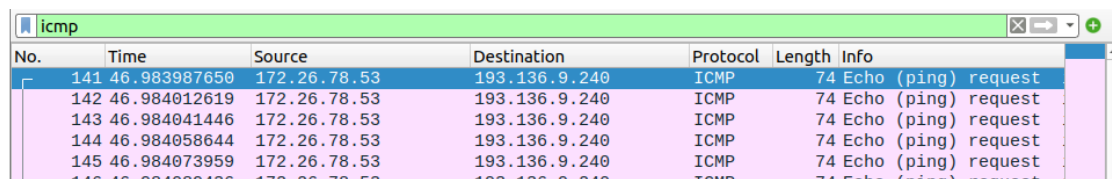
"Pretende-se agora analisar a fragmentação de pacotes IP. Usando o Wireshark, capture e observe o tráfego gerado depois do tamanho de pacote ter sido definido para (3xy0) bytes, em que xy é o número do seu grupo de trabalho (e.g., o grupo PL19 deve usar um tamanho de pacote de 3190 bytes). De modo a poder visualizar os fragmentos, aceda a Edit -> Preferences -> Protocols e em IPv4 desative a opção "Reassemble fragmented IPv4 datagrams". Nota: Como alternativa para geração do tráfego pode usar o comando ping <opção> <bytes> marco.uminho.pt, onde a opção -l (Windows) ou -s (Linux, Mac) permite definir o número de bytes enviados no campo de dados do pacote ICMP. Documente e justifique todas as respostas às seguintes alíneas:"

```
jose02@joseMachine:~$ ping -s 3740 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 3740(3768) bytes of data.
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=1 ttl=61 time=2.52 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=2 ttl=61 time=7.26 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=3 ttl=61 time=3.27 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=4 ttl=61 time=25.1 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=5 ttl=61 time=2.99 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=6 ttl=61 time=3.80 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=7 ttl=61 time=3.03 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=8 ttl=61 time=3.42 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=9 ttl=61 time=3.81 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=10 ttl=61 time=6.75 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=11 ttl=61 time=4.96 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=12 ttl=61 time=5.48 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=13 ttl=61 time=5.45 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=14 ttl=61 time=2.97 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=15 ttl=61 time=3.79 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=16 ttl=61 time=4.97 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=17 ttl=61 time=4.51 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=18 ttl=61 time=14.0 ms
3748 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=19 ttl=61 time=7.54 ms
```

Figura 1.12: Resultados do comando ping -s 3740 marco.uminho.pt

1.3.1 a

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



No.	Time	Source	Destination	Protocol	Length	Info
141	46.983987650	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
142	46.984012619	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
143	46.984041446	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
144	46.984058644	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
145	46.984073959	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request
146	46.984088436	172.26.78.53	193.136.9.240	ICMP	74	Echo (ping) request

Figura 1.13: Primeira mensagem ICMP

R: A necessidade de fragmentar o pacote inicial ocorre devido ao *MTU (Maximum Transfer Unit)* ser de apenas 1500 bytes, sendo que o pacote enviado possui um tamanho de 3740 bytes, ocorreu assim, o processo de fragmentação do mesmo. Neste caso, o pacote foi dividido em 3 pacotes.

1.3.2 b

"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?"

```
.... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x1198 (4504)
  001. .... = Flags: 0x1, 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: 64
  Protocol: ICMP (1)
  Header Checksum: 0x7dc1 [validation disabled]
  [Header checksum status: Unverified]
```

Figura 1.14: Informação relativa ao primeiro segmento

R: Como é possível ver na imagem, a flag **More Fragments** do cabeçalho encontra-se como **Set**, e com o valor "1" é possível perceber que ocorreu fragmentação. Assim, como o **Fragment Offset** é 0, sabemos que este se trata do primeiro fragmento de tamanho é de 1500, em que 1480 bytes são de dados e 20 bytes de header.

1.3.3 c

"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?"

```
.... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0x1198 (4504)
  001. .... = Flags: 0x1, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
    ...0 0000 1011 1001 = Fragment Offset: 1480
```

Figura 1.15: Informação relativa ao segundo fragmento

R: Sendo que o fragmento possui um **Offset** de 1480, é notável que este se trata do segundo fragmento, pois já assume o tamanho do fragmento inicial de 1480 bytes. É possível ainda observar que ainda irá haver mais fragmentos após este, devido à flag **More Fragments** encontrar-se como **Set** e ter o valor de 1.

1.3.4 d

"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."

R: O pacote será dividido em 3 fragmentos sendo que o primeiro fragmento será de 1500 bytes (1472 bytes de dados, 8 bytes de cabeçalho ICMP, 20 bytes de header), o segundo fragmento terá 1500 bytes (1480 bytes de dados, 20 bytes de header) e o terceiro fragmento terá 808 bytes (788 bytes de dados, 20 bytes de header).

O número de fragmentos e bytes são iguais ao representados no Wireshark.

1.3.5 e

"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."

R: Vamos recapitular as fragmentações feitas. Para o primeiro pacote, iniciado no 'Fragment Offset' 0, são guardados 1480 bytes de dados e 20 bytes para o cabeçalho IP. Foi precisa fragmentação, pois faltavam ainda ser enviados 2260 bytes de dados.

Para o segundo pacote, iniciado no '*Fragment Offset*' 1480, foram guardados novamente 1480 *bytes* de dados e 20 *bytes* para o cabeçalho *IP*. Foi precisa fragmentação novamente, pois ainda restavam 780 *bytes* de dados para serem enviados.

O terceiro pacote inicia-se no '*Fragment Offset*' 2960 (1480+1480), sendo reservados 780 *bytes* para dados e 20 *bytes* para o cabeçalho *IP*. Não haverá mais fragmentação a partir deste pacote, uma vez que (1480+1480+780) *bytes* = 3740 *bytes*, indicando que o total de dados já foi enviado.

Tal pode ser visto esquematicamente pela foto abaixo.

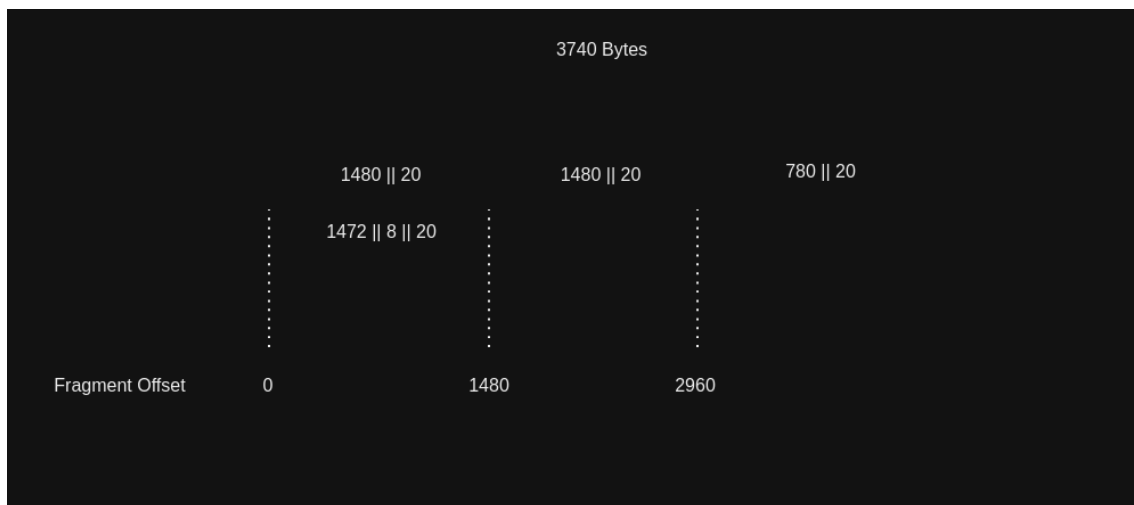


Figura 1.16: Representação gráfica da fragmentação feita com indicação dos respetivos '*Fragment Offsets*'

Abaixo, vemos que o '*Fragment Offset*' encontra-se em 2960, indicando que estamos a lidar com o terceiro pacote. Descartando esta informação já antecipadamente sabida, o *WireShark* permite-nos saber isso pelo seguinte: apesar do campo '*Don't fragment*' permitir a fragmentação, não há essa necessidade, pois com a indicação de '*More Fragments*' = *Not set* percebe-se que não haverá mais fragmentação a ser feita.

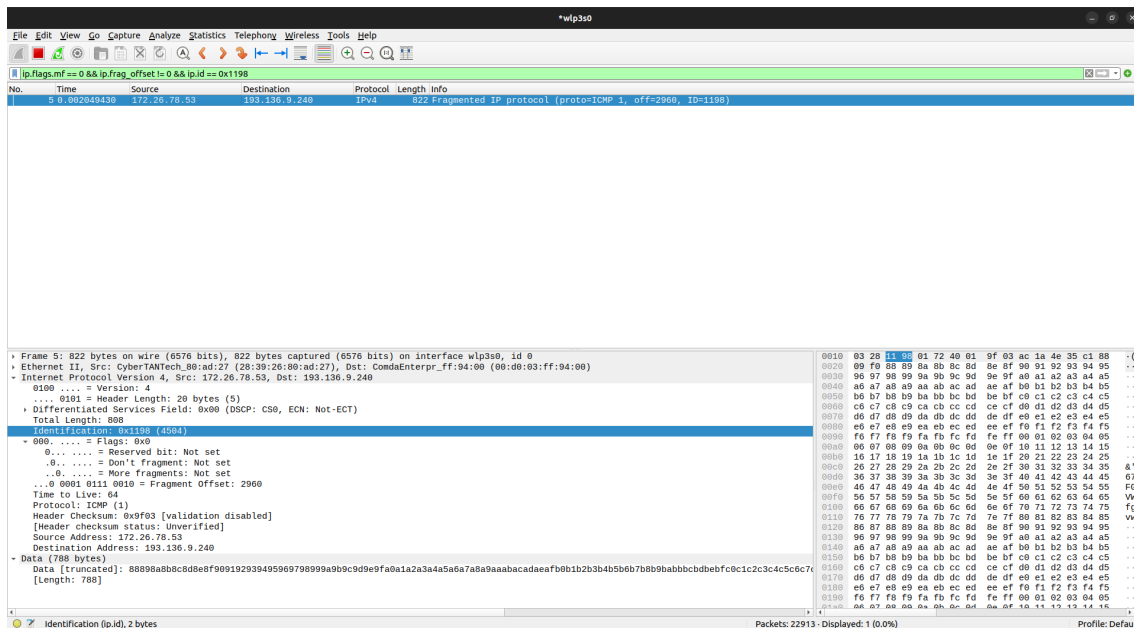


Figura 1.17: Informação relativa ao terceiro fragmento

1.3.6 f

"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?"

R: O equipamento em que ocorre a reconstrução do pacote é a interface de destino, neste caso, o equipamento de **IP 193.136.9.240**. Com isto, é necessário todos os pacotes para realizar uma reconstrução confiável, assim como todos os cabeçalhos dos mesmos para que seja possível identificar a ordem de reconstrução.

1.3.7 g

"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."

R: Nos cabeçalhos *IP*, os campos que variam entre fragmentos incluem as *flags* 'Fragment Offset', 'More Fragments' e 'Total Length'.

- O tamanho total do último fragmento é diferente dos dois anteriores, que possuem o mesmo tamanho.

- O '*Fragment Offset*' do primeiro fragmento é 0, o '*Fragment Offset*' do segundo é igual ao tamanho do primeiro e o '*Fragment Offset*' do último é o tamanho dos dados dos pacotes anteriores incrementado.
- A flag '*More Fragments*' está definida como 1 nos dois primeiros fragmentos e como 0 no último. Tal informação é crucial para estabelecer a ordem de reconstrução do datagrama original e determinar quando todos os fragmentos de um datagrama são recebidos na totalidade.

1.3.8 h

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

R: Apenas o primeiro fragmento de cada pacote é identificado como *ICMP*, devido ao facto de este conter o cabeçalho original, enquanto os fragmentos seguintes apenas contêm informações sobre a fragmentação do pacote e as suas ordens.

1.3.9 i

"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?"

R: O tamanho do datagrama é comparado com o valor do *MTU* de 1500 bytes. Aumentar e diminuir o valor do *MTU* pode causar efeitos significativos na própria rede. Caso seja aumentado o tamanho do datagrama, a eficiência da própria rede irá aumentar devido a um número menor de datagramas serão fragmentados, porém, é capaz de criar problemas com interfaces que não suportam um *MTU* maior. Um *MTU* maior também pode causar um tempo de transmissão maior ao esperado, sendo que apesar de obter menos fragmentos, estes serão maiores que o normal. Contudo, diminuir o tamanho do datagrama, resultará numa sobrecarga sobre a fragmentação dos datagramas maiores, resultando num desempenho de rede mais lento, porém, esta diminuição poderá evitar problemas de fragmentação e perda de pacotes em redes que não suportam transmissões de maiores valores.

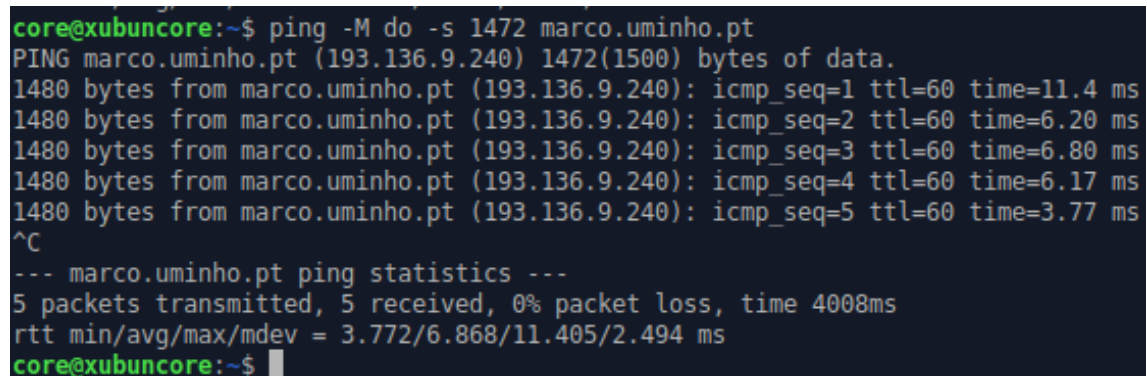
1.3.10 j

"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."

R: O valor máximo em *bytes* do pacote que pode ser transmitido sem fragmentação é de 1472. Isto acontece, pois a *MTU* reserva 1500 *bytes*. Estando reservados 20 *bytes* para o cabeçalho *IP* e 8 *bytes* para o cabeçalho *ICMP* (a mensagem *ICMP* envia uma atualização de *status*), então restam apenas 1472 *bytes* para que o pacote transmita sem fragmentação.

Expressão que mostra o que foi apresentado [(1500-20-8) bytes = 1472 bytes]

Concluindo, o valor máximo de *SIZE* é de 1472 *bytes*.



```
core@xubuncore:~$ 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=60 time=11.4 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=2 ttl=60 time=6.20 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=3 ttl=60 time=6.80 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=4 ttl=60 time=6.17 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=5 ttl=60 time=3.77 ms
^C
--- marco.uminho.pt ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 3.772/6.868/11.405/2.494 ms
core@xubuncore:~$
```

Figura 1.18: Ping efetuado que mostra o valor máximo sem fragmentação

2 Parte 2

2.1 Questão 1

"Com os avanços da Inteligência Artificial, D. Afonso Henriques termina todas as suas tarefas mais cedo e vê-se com algum tempo livre. Decide então fazer remodelações no reino:"

2.1.1 a

"De modo a garantir uma posição estrategicamente mais vantajosa e ter casa de férias para relaxar entre batalhas, ordena a construção de um segundo Castelo, em Braga. Não tendo qualquer queixa do serviço prestado, recorre aos serviços do ISP ReiDaNet, que já utiliza no condado, para ter acesso à rede no segundo Castelo. O ISP atribuiu-lhe o endereço de rede IP 172.XX.33.128/26 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 12 ou mais hosts. Assuma que todos os endereços de sub-redes são utilizáveis."

R: O endereço de rede IP atribuído pelo ISP corresponde a 172.74.33.128/26. Para que haja o estabelecimento de pelo menos 3 redes, terão de ser reservados 2 bits, originando sub-redes em que o bit número 27 e número 28 são respetivamente os seguintes 00, 01, 10, 11 ($2^2 = 4 > 3$), seguidos de zeros (bit número 29, 30, 31, 32 com 0).

As sub-redes originadas são então as seguintes:

- 172.74.33.128/28
- 172.74.33.144/28
- 172.74.33.160/28
- 172.74.33.176/28

Os primeiros 8 bits de endereço IP são dedicados à identificação de rede. Assim, os restantes, estarão 4 bits reservados para identificar hosts dessa rede (os bits número 29, 30, 31, 32). O número de hosts para cada uma das redes será portanto $2^4 - 2 = 14 > 12$. É feita a subtração por 2, devido ao descarte do host da própria rede e ao broadcast.

2.1.2 b

"Ligue um novo host Castelo2 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 dispositivos do Condado Portucalense."

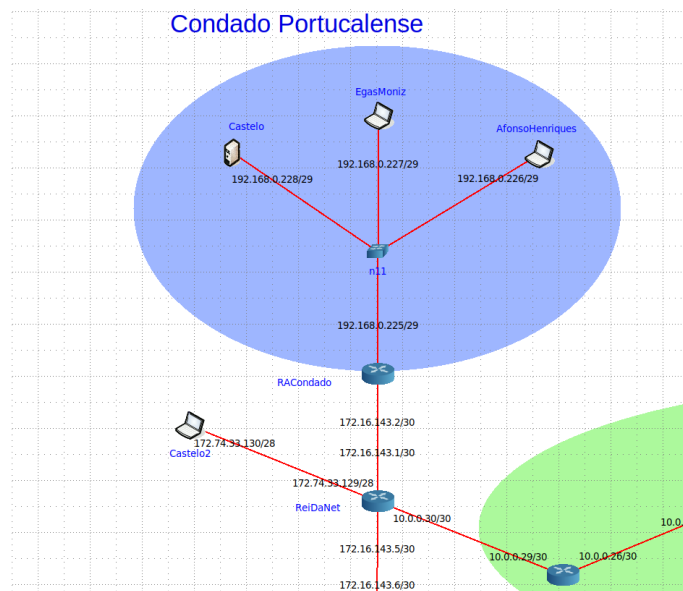


Figura 2.1: Representação da ligação de Castelo2 diretamente ao router ReiDaNet

```
vcmd
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# ping 192.168.0.226
PING 192.168.0.226 (192.168.0.226) 56(84) bytes of data:
64 bytes from 192.168.0.226: icmp_seq=1 ttl=62 time=0.037 ms
64 bytes from 192.168.0.226: icmp_seq=2 ttl=62 time=0.047 ms
64 bytes from 192.168.0.226: icmp_seq=3 ttl=62 time=0.046 ms
^C
--- 192.168.0.226 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 0.037/0.043/0.047/0.004 ms
root@Castelo2:/tmp/pycore.33405/Castelo2.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=62 time=0.118 ms
64 bytes from 192.168.0.227: icmp_seq=2 ttl=62 time=0.070 ms
64 bytes from 192.168.0.227: icmp_seq=3 ttl=62 time=0.328 ms
^C
--- 192.168.0.227 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2053ms
rtt min/avg/max/mdev = 0.070/0.172/0.328/0.112 ms
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# ping 192.168.0.228
PING 192.168.0.228 (192.168.0.228) 56(84) bytes of data:
64 bytes from 192.168.0.228: icmp_seq=1 ttl=62 time=0.063 ms
64 bytes from 192.168.0.228: icmp_seq=2 ttl=62 time=0.051 ms
64 bytes from 192.168.0.228: icmp_seq=3 ttl=62 time=0.050 ms
^C
--- 192.168.0.228 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2042ms
rtt min/avg/max/mdev = 0.050/0.054/0.063/0.005 ms
root@Castelo2:/tmp/pycore.33405/Castelo2.conf#
```

Figura 2.2: Análise da conectividade com os dispositivos do Condado Portucalense

R: Com a introdução de um novo host com o seguinte IP, **172.74.33.130/28**, é possível

verificar a conectividade com todos os dispositivos pertencentes ao Condado Portucalense.

2.1.3 c

"Não estando satisfeito com a decoração deste novo Castelo, opta por eliminar a sua rota default. Adicione as rotas necessárias para que o Castelo2 continue a ter acesso ao Condado Portucalense e à rede Institucional. Mostre que a conectividade é restabelecida, assim como a tabela de encaminhamento resultante. Explícite ainda a utilidade de uma rota default."

```
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0        172.74.33.129  0.0.0.0         UG      0 0        0 eth0
172.74.33.128  0.0.0.0        255.255.255.240 U        0 0        0 eth0
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# ping 192.168.0.226
PING 192.168.0.226 (192.168.0.226) 56(84) bytes of data:
64 bytes from 192.168.0.226: icmp_seq=1 ttl=62 time=0.078 ms
64 bytes from 192.168.0.226: icmp_seq=2 ttl=62 time=0.054 ms
64 bytes from 192.168.0.226: icmp_seq=3 ttl=62 time=0.057 ms
64 bytes from 192.168.0.226: icmp_seq=4 ttl=62 time=0.054 ms
64 bytes from 192.168.0.226: icmp_seq=5 ttl=62 time=0.053 ms
^C
--- 192.168.0.226 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4101ms
rtt min/avg/max/mdev = 0.053/0.059/0.078/0.009 ms
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# route add -net 192.168.0.224
SIOCADDRT: Invalid argument
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# route add -net 192.168.0.224 netmask 255.255.255.248 gw 172.74.33.129
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# ping 192.168.0.226
PING 192.168.0.226 (192.168.0.226) 56(84) bytes of data:
64 bytes from 192.168.0.226: icmp_seq=1 ttl=62 time=0.041 ms
64 bytes from 192.168.0.226: icmp_seq=2 ttl=62 time=0.068 ms
64 bytes from 192.168.0.226: icmp_seq=3 ttl=62 time=0.052 ms
^C
--- 192.168.0.226 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2041ms
rtt min/avg/max/mdev = 0.041/0.053/0.068/0.011 ms
root@Castelo2:/tmp/pycore.33405/Castelo2.conf# route add -net 192.168.0.224 netmask 255.255.255.248 gw 172.74.33.129
```

Figura 2.3: Tabela de encaminhamento

R: Com a rota *default*, é relacionada a capacidade do próprio *router* de encaminhar o tráfego de rede da forma mais eficiente e segura possível. Caso a mesma não exista, não serão enviados os pacotes de redes para outros destinos ainda desconhecidos, recorrendo ao descartamento dos mesmos.

2.2 Questão 2

"D.Afonso Henriques quer enviar fotos do novo Castelo à sua mãe, D.Teresa, mas está a ter alguns problemas de comunicação. Este alega que o problema deverá estar no dispositivo de D.Teresa, uma vez que no dia anterior conseguiu enviar a sua declaração do IRS para o portal das finanças, e não tem qualquer problema em ver as suas séries favoritas, disponíveis na rede de conteúdos."

2.2.1 a

"Confirme, através do comando ping, que AfonsoHenriques tem efetivamente conectividade com o servidor *Financas* e com os servidores da *CDN*."

R: Com as imagens abaixo, é possível reconhecer a conectividade com os servidores *Financas* e *CDN*.

```
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf#  
<ycore.33405/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=0.120 ms  
64 bytes from 192.168.0.250: icmp_seq=2 ttl=61 time=0.060 ms  
64 bytes from 192.168.0.250: icmp_seq=3 ttl=61 time=0.062 ms  
64 bytes from 192.168.0.250: icmp_seq=4 ttl=61 time=0.064 ms  
64 bytes from 192.168.0.250: icmp_seq=5 ttl=61 time=0.061 ms  
^C  
--- 192.168.0.250 ping statistics ---  
5 packets transmitted, 5 received, 0% packet loss, time 4097ms  
rtt min/avg/max/mdev = 0.060/0.073/0.120/0.023 ms  
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf# financas
```

Figura 2.4: Conectividade com o servidor 'Finanças'

```
<core.33405/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=0.191 ms  
64 bytes from 192.168.0.204: icmp_seq=2 ttl=55 time=0.107 ms  
64 bytes from 192.168.0.204: icmp_seq=3 ttl=55 time=0.104 ms  
64 bytes from 192.168.0.204: icmp_seq=4 ttl=55 time=0.101 ms  
64 bytes from 192.168.0.204: icmp_seq=5 ttl=55 time=0.119 ms  
^C  
--- 192.168.0.204 ping statistics ---  
5 packets transmitted, 5 received, 0% packet loss, time 4089ms  
rtt min/avg/max/mdev = 0.101/0.124/0.191/0.033 ms  
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf# hbo
```

Figura 2.5: Conectividade com o servidor 'HBO'


```

<core.33405/AfonsoHenriques.conf# ping 192.168.0.210
PING 192.168.0.210 (192.168.0.210) 56(84) bytes of data.
64 bytes from 192.168.0.210: icmp_seq=1 ttl=55 time=0.280 ms
64 bytes from 192.168.0.210: icmp_seq=2 ttl=55 time=0.099 ms
64 bytes from 192.168.0.210: icmp_seq=3 ttl=55 time=0.120 ms
64 bytes from 192.168.0.210: icmp_seq=4 ttl=55 time=0.098 ms
64 bytes from 192.168.0.210: icmp_seq=5 ttl=55 time=0.097 ms
^C
--- 192.168.0.210 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4081ms
rtt min/avg/max/mdev = 0.097/0.138/0.280/0.071 ms
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf# iTunes

```

Figura 2.6: Conectividade com o servidor '*iTunes*'

```

<core.33405/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=0.087 ms
64 bytes from 192.168.0.203: icmp_seq=2 ttl=55 time=0.110 ms
64 bytes from 192.168.0.203: icmp_seq=3 ttl=55 time=0.099 ms
64 bytes from 192.168.0.203: icmp_seq=4 ttl=55 time=0.099 ms
64 bytes from 192.168.0.203: icmp_seq=5 ttl=55 time=0.100 ms
^C
--- 192.168.0.203 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4101ms
rtt min/avg/max/mdev = 0.087/0.099/0.110/0.007 ms
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf# netflix

```

Figura 2.7: Conectividade com o servidor '*Netflix*'

```

<core.33405/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=0.204 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=55 time=0.103 ms
64 bytes from 192.168.0.218: icmp_seq=3 ttl=55 time=0.102 ms
64 bytes from 192.168.0.218: icmp_seq=4 ttl=55 time=0.105 ms
64 bytes from 192.168.0.218: icmp_seq=5 ttl=55 time=0.108 ms
^C
--- 192.168.0.218 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4074ms
rtt min/avg/max/mdev = 0.102/0.124/0.204/0.039 ms
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf# spotify

```

Figura 2.8: Conectividade com o servidor '*Spotify*'

```

<ycore.33405/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=0.507 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.170 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.120 ms
64 bytes from 192.168.0.202: icmp_seq=4 ttl=55 time=0.106 ms
64 bytes from 192.168.0.202: icmp_seq=5 ttl=55 time=0.172 ms
^C
--- 192.168.0.202 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4091ms
rtt min/avg/max/mdev = 0.106/0.215/0.507/0.148 ms
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf# youtube

```

Figura 2.9: Conectividade com o servidor 'Youtube'

2.2.2 b

"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."

R: Através das imagens das tabelas de encaminhamento de AfonsoHenriques e de Teresa, é possível observar que não existe nenhum problema referente às mesmas.

- Na tabela de AfonsoHenriques, é possível observar na primeira entrada que o destino padrão é 0.0.0.0 e que o *gateway* é 192.168.0.225, com isto, é possível reconhecer que caso não consiga encaminhar um pacote para um certo destino marcado, este irá ser entregue na rede destino. A *genmask* de valor 0.0.0.0 prova os bits usados para verificar se o pacote pertence à entrada correspondente. A segunda entrada possui um endereço de destino de 192.168.0.224 e uma *genmask* de valor 255.255.255.248, que permite a entrada de todos os endereços IP na rede disponível.
- Na tabela de Teresa, é possível observar na primeira entrada que o destino padrão é 0.0.0.0 e que o *gateway* é 192.168.0.193, com isto, é possível reconhecer que caso não consiga encaminhar um pacote para um certo destino, este irá entregar o mesmo na rede destino. A *genmask* de valor 0.0.0.0 demonstra os bits usados para verificar se o pacote pertence à entrada correspondente. A segunda entrada possui um endereço de destino de 192.168.0.192 e uma *genmask* de valor 255.255.255.248, que permite a entrada de todos os endereços IP na rede disponível.

As tabelas de encaminhamento apresentam-se abaixo.

```

root@AfonsoHenriques:/tmp/pycore.33405/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
root@AfonsoHenriques:/tmp/pycore.33405/AfonsoHenriques.conf# █

```

Figura 2.10: Tabela de encaminhamento do dispositivo AfonsoHenriques

```

root@Teresa:/tmp/pycore.33405/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.33405/Teresa.conf# █

```

Figura 2.11: Tabela de encaminhamento do dispositivo Teresa

2.2.3 c

"Análise 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."

R: Este exercício, devido à sua complexidade será explicado em vários passos:

- Devido a uma falta de conexão entre AfonsoHenriques e Teresa, foi adicionado um caminho em n5, para permitir contínua conexão.

```

<5.conf# route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.25
root@n5:/tmp/pycore.45035/n5.conf# █

```

Figura 2.12: Comando *add* no n5

- De seguida, foi removido um caminho em n2 que causava problemas de loop na rede.

```

<2.conf# route del -net 192.168.0.194 netmask 255.255.255.254
root@n2:/tmp/pycore.45035/n2.conf# █

```

Figura 2.13: Comando *del* no n2

- Foi então apagado um caminho em n1 com problemas de conexão com os seguintes, em que levava o caminho de rede para o anterior, e criamos assim uma nova conexão para o n1 que permitia que o mesmo continuasse o seu caminho.

```
<1.conf# route del -net 192.168.0.192 netmask 255.255.255.248
<1.conf# route add -net 192.168.0.192 gw 10.0.0.9 netmask 255.255.255.248
root@n1:/tmp/pycore.45035/n1.conf#
```

Figura 2.14: Comandos *del* e *add* no n1

- É possível observar nas seguintes figuras o progresso da conexão entre AfonsoHenriques e Teresa.

```
<ycore.45035/AfonsoHenriques.conf# traceroute -I 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) 0.029 ms 0.005 ms 0.003 ms
 2 172.16.143.1 (172.16.143.1) 0.014 ms 0.005 ms 0.005 ms
 3 10.0.0.29 (10.0.0.29) 0.014 ms !N 0.007 ms !N *
```

Figura 2.15: Comando *traceroute* no AfonsoHenriques 1º passo

```
<5/AfonsoHenriques.conf# traceroute -I 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) 0.033 ms 0.005 ms 0.003 ms
 2 172.16.143.1 (172.16.143.1) 0.013 ms 0.005 ms 0.005 ms
 3 10.0.0.29 (10.0.0.29) 0.014 ms 0.006 ms 0.006 ms
 4 10.0.0.25 (10.0.0.25) 0.017 ms 0.008 ms 0.009 ms
 5 10.0.0.13 (10.0.0.13) 0.030 ms 0.015 ms 0.028 ms
 6 10.0.0.17 (10.0.0.17) 0.093 ms 0.024 ms 0.011 ms
 7 10.0.0.5 (10.0.0.5) 0.056 ms 0.017 ms 0.014 ms
 8 10.0.0.1 (10.0.0.1) 0.052 ms 0.018 ms 0.027 ms^C
root@AfonsoHenriques:/tmp/pycore.45035/AfonsoHenriques.conf#
```

Figura 2.16: Comando *traceroute* no AfonsoHenriques 4º passo

2.2.4 d

"Uma vez que o core da rede esteja a encaminhar corretamente os pacotes enviados por AfonsoHenriques, confira com o Wireshark se estes são recebidos por Teresa."

```
/RAGaliza.conf# route add -net 192.168.0.224 gw 172.16.142.1 netmask 255.255.255.248
```

Figura 2.17: Add do route de regresso em 'RAGaliza'

```
root@RAGaliza:/tmp/pycore.45035/RAGaliza.conf# netstat -rn
Kernel IP routing table
Destination        Gateway           Genmask          Flags        MSS Window  irtt Iface
10.0.0.0            172.16.142.1     255.255.255.252 UG            0 0         0 eth0
10.0.0.4            172.16.142.1     255.255.255.252 UG            0 0         0 eth0
10.0.0.8            172.16.142.1     255.255.255.252 UG            0 0         0 eth0
10.0.0.12           172.16.142.1     255.255.255.252 UG            0 0         0 eth0
10.0.0.16           172.16.142.1     255.255.255.252 UG            0 0         0 eth0
10.0.0.20           172.16.142.1     255.255.255.252 UG            0 0         0 eth0
10.0.0.24           172.16.142.1     255.255.255.252 UG            0 0         0 eth0
10.0.0.28           172.16.142.1     255.255.255.252 UG            0 0         0 eth0
172.0.0.0           172.16.142.1     255.0.0.0        UG            0 0         0 eth0
172.16.142.0        0.0.0.0           255.255.255.252 U            0 0         0 eth0
172.16.142.4        172.16.142.1     255.255.255.252 UG            0 0         0 eth0
172.16.143.0        172.16.142.1     255.255.255.252 UG            0 0         0 eth0
172.16.143.4        172.16.142.1     255.255.255.252 UG            0 0         0 eth0
192.168.0.192       0.0.0.0           255.255.255.248 U            0 0         0 eth1
192.168.0.200       172.16.142.1     255.255.255.248 UG            0 0         0 eth0
192.168.0.208       172.16.142.1     255.255.255.248 UG            0 0         0 eth0
192.168.0.216       172.16.142.1     255.255.255.248 UG            0 0         0 eth0
192.168.0.224       172.16.142.1     255.255.255.248 UG            0 0         0 eth0
192.168.0.232       172.16.142.1     255.255.255.248 UG            0 0         0 eth0
192.168.0.240       172.16.142.1     255.255.255.248 UG            0 0         0 eth0
192.168.0.248       172.16.142.1     255.255.255.248 UG            0 0         0 eth0
root@RAGaliza:/tmp/pycore.45035/RAGaliza.conf#
```

Figura 2.18: Netstat de RAGaliza

1442	16587.871929..	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=0x0033, seq=1/256, ttl=55 (reply in 1443)
1443	16587.872006..	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=0x0033, seq=1/256, ttl=64 (request in 1442)
1444	16588.116172..	192.168.0.193	224.0.0.5	OSPF	78 Hello Packet	
1445	16588.694453..	fe80::200:ff:feaa:10	ff02::5	OSPF	90 Hello Packet	
1446	16588.873048..	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=0x0033, seq=2/512, ttl=55 (reply in 1447)
1447	16588.873050..	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=0x0033, seq=2/512, ttl=64 (request in 1446)
1448	16589.899055..	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=0x0033, seq=3/768, ttl=55 (reply in 1449)
1449	16589.899065..	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=0x0033, seq=3/768, ttl=64 (request in 1448)
1450	16590.116644..	192.168.0.193	224.0.0.5	OSPF	78 Hello Packet	
1451	16590.923611..	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=0x0033, seq=4/1024, ttl=55 (reply in 1452)
1452	16590.923657..	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=0x0033, seq=4/1024, ttl=64 (request in 1451)
1453	16591.947100..	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=0x0033, seq=5/1280, ttl=55 (reply in 1454)
1454	16591.947109..	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=0x0033, seq=5/1280, ttl=64 (request in 1453)
1455	16592.117723..	192.168.0.193	224.0.0.5	OSPF	78 Hello Packet	
1456	16592.971426..	00:00:00:aa:00:13	00:00:00:aa:00:10	ARP	42 Who has 192.168.0.193? Tell 192.168.0.194	
1457	16592.971536..	00:00:00:aa:00:10	00:00:00:aa:00:13	ARP	42 Who has 192.168.0.194? Tell 192.168.0.193	
1458	16592.971728..	00:00:00:aa:00:10	00:00:00:aa:00:13	ARP	42 192.168.0.193 is at 00:00:00:aa:00:10	
1459	16592.971689..	00:00:00:aa:00:13	00:00:00:aa:00:10	ARP	42 192.168.0.194 is at 00:00:00:aa:00:13	
1460	16592.971767..	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=0x0033, seq=6/1536, ttl=55 (reply in 1461)
1461	16592.971772..	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=0x0033, seq=6/1536, ttl=64 (request in 1460)
1462	16593.995598..	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=0x0033, seq=7/1792, ttl=55 (reply in 1463)
1463	16593.995607..	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=0x0033, seq=7/1792, ttl=64 (request in 1462)

Figura 2.19: Imagem retirada do Wireshark que mostra os pacotes

i)

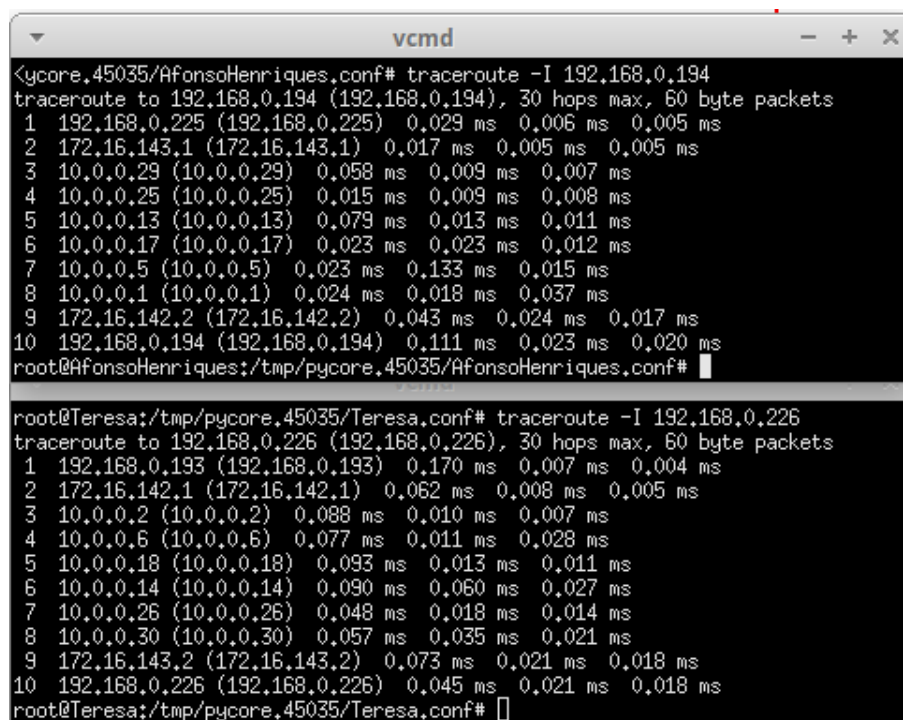
"Em caso afirmativo, porque é que continua a não existir conectividade entre D.Teresa e D.Afonso Henriques? Efetue as alterações necessárias para garantir que a conectividade é

restabelecida e o confronto entre os dois é evitado.”

R: Pelo *Wireshark* sabemos que o *host* Teresa está a receber os pacotes enviados pelo *host* AfonsoHenriques pelo *traceroute*. Porém, AfonsoHenriques não obtém com êxito os pacotes de resposta que Teresa envia. Conclui-se que em RAGaliza, não está a ser feito o encaminhamento dos pacotes de volta, daí a adição da entrada à tabela de encaminhamento.

ii)

“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*.”



```
vcmd
<ycore.45035/AfonsoHenriques.conf# traceroute -I 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) 0.029 ms 0.006 ms 0.005 ms
 2 172.16.143.1 (172.16.143.1) 0.017 ms 0.005 ms 0.005 ms
 3 10.0.0.29 (10.0.0.29) 0.058 ms 0.009 ms 0.007 ms
 4 10.0.0.25 (10.0.0.25) 0.015 ms 0.009 ms 0.008 ms
 5 10.0.0.13 (10.0.0.13) 0.079 ms 0.013 ms 0.011 ms
 6 10.0.0.17 (10.0.0.17) 0.023 ms 0.023 ms 0.012 ms
 7 10.0.0.5 (10.0.0.5) 0.023 ms 0.133 ms 0.015 ms
 8 10.0.0.1 (10.0.0.1) 0.024 ms 0.018 ms 0.037 ms
 9 172.16.142.2 (172.16.142.2) 0.043 ms 0.024 ms 0.017 ms
10 192.168.0.194 (192.168.0.194) 0.111 ms 0.023 ms 0.020 ms
root@AfonsoHenriques:/tmp/pycore.45035/AfonsoHenriques.conf#

root@Teresa:/tmp/pycore.45035/Teresa.conf# traceroute -I 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) 0.170 ms 0.007 ms 0.004 ms
 2 172.16.142.1 (172.16.142.1) 0.062 ms 0.008 ms 0.005 ms
 3 10.0.0.2 (10.0.0.2) 0.088 ms 0.010 ms 0.007 ms
 4 10.0.0.6 (10.0.0.6) 0.077 ms 0.011 ms 0.028 ms
 5 10.0.0.18 (10.0.0.18) 0.093 ms 0.013 ms 0.011 ms
 6 10.0.0.14 (10.0.0.14) 0.090 ms 0.060 ms 0.027 ms
 7 10.0.0.26 (10.0.0.26) 0.048 ms 0.018 ms 0.014 ms
 8 10.0.0.30 (10.0.0.30) 0.057 ms 0.035 ms 0.021 ms
 9 172.16.143.2 (172.16.143.2) 0.073 ms 0.021 ms 0.018 ms
10 192.168.0.226 (192.168.0.226) 0.045 ms 0.021 ms 0.018 ms
root@Teresa:/tmp/pycore.45035/Teresa.conf#
```

Figura 2.20: *Traceroute* dos caminhos usando o comando *traceroute*

R: Podemos observar que os trajetos de ambos sejam semelhantes, em que difere apenas no caminho entre n1 e n4. Enquanto AfonsoHenriques toma o caminho por n1, Teresa toma o caminho por n4.


```

root@n3:/tmp/pycore.45035/n3.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags       MSS  Window  irtt  Iface
10.0.0.0          10.0.0.5        255.255.255.252 UG           0    0         0    eth2
10.0.0.4          0.0.0.0         255.255.255.252 U            0    0         0    eth2
10.0.0.8          0.0.0.0         255.255.255.252 U            0    0         0    eth0
10.0.0.12         10.0.0.10       255.255.255.252 UG           0    0         0    eth0
10.0.0.16         0.0.0.0         255.255.255.252 U            0    0         0    eth1
10.0.0.20         10.0.0.18       255.255.255.252 UG           0    0         0    eth1
10.0.0.24         10.0.0.18       255.255.255.252 UG           0    0         0    eth1
10.0.0.28         10.0.0.10       255.255.255.252 UG           0    0         0    eth0
172.0.0.0         10.0.0.10       255.0.0.0       UG           0    0         0    eth0
172.16.142.0      10.0.0.5        255.255.255.252 UG           0    0         0    eth2
172.16.142.4      10.0.0.5        255.255.255.252 UG           0    0         0    eth2
172.16.143.0      10.0.0.18       255.255.255.252 UG           0    0         0    eth1
172.16.143.4      10.0.0.10       255.255.255.252 UG           0    0         0    eth0
192.168.0.192     10.0.0.5        255.255.255.248 UG           0    0         0    eth2
192.168.0.192     10.0.0.18       255.255.255.240 UG           0    0         0    eth1
192.168.0.200     10.0.0.5        255.255.255.248 UG           0    0         0    eth2
192.168.0.208     10.0.0.5        255.255.255.248 UG           0    0         0    eth2
192.168.0.216     10.0.0.5        255.255.255.248 UG           0    0         0    eth2
192.168.0.224     10.0.0.18       255.255.255.248 UG           0    0         0    eth1
192.168.0.232     10.0.0.10       255.255.255.248 UG           0    0         0    eth0
192.168.0.240     10.0.0.10       255.255.255.248 UG           0    0         0    eth0
192.168.0.248     10.0.0.10       255.255.255.248 UG           0    0         0    eth0
root@n3:/tmp/pycore.45035/n3.conf# traceroute 192.168.0.192
traceroute to 192.168.0.192 (192.168.0.192), 30 hops max, 60 byte packets

```

Figura 2.23: Traceroute e netstat no sentido n3-Galiza

2.2.6 f

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

R: É possível verificar que todos os 4 polos disponibilizados, assim como os que são usados nas redes/ISPs são endereços privados devido a estes localizarem-se no raio de endereçamento privado dos mesmos, sendo os seguintes:

- 10.0.0.0-10.255.255.255 (para as core)
- 192.168.0.0-192.168.255.255 (para os 4 polos)
- 172.16.0.0-172.31.255.255 (para as redes/ISPs)

Os endereços privados correspondem corretamente às necessidades de conectividade interna de redes quer privadas quer locais, sendo indispensável o recurso a endereços públicos, assegurando

o controlo dos canais de comunicação.

2.2.7 g

"Os switches localizados em cada um dos polos têm um endereço IP atribuído? Porquê?"

R: De acordo com as suas definições, os switches são dispositivos que apenas operam nas segundas camadas da rede, recorrendo assim a endereços MAC para a sua funcionalidade na própria rede. Resumindo, não existe necessidade de as mesmas funcionarem em conjunto com endereços IP.

2.3 Questão 3

"Ao ver as fotos no CondadoGram, D. Teresa não ficou convencida com as novas alterações e ordena que Afonso Henriques vá arrumar o castelo. Inconformado, este decide planejar um novo ataque, mas constata que o seu exército não só perde bastante tempo a decidir que direção tomar a cada salto como, por vezes, inclusivamente se perde."

2.3.1 a

"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."

```

root@n6:/tmp/pycore.45035/n6.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags        MSS Window  irtt Iface
10.0.0.0         0.0.0.0         255.255.255.252 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        255.255.255.248 UG           0 0        0 eth0
192.168.0.200    10.0.0.1        255.255.255.248 UG           0 0        0 eth0
192.168.0.208    10.0.0.1        255.255.255.248 UG           0 0        0 eth0
192.168.0.216    10.0.0.1        255.255.255.248 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
root@n6:/tmp/pycore.45035/n6.conf# traceroute 192.168.0.203
traceroute to 192.168.0.203 (192.168.0.203), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.023 ms 0.004 ms 0.004 ms
 2 172.16.142.6 (172.16.142.6) 0.013 ms 0.006 ms 0.005 ms
 3 192.168.0.203 (192.168.0.203) 0.012 ms 0.007 ms 0.006 ms
root@n6:/tmp/pycore.45035/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) 0.023 ms 0.005 ms 0.003 ms
 2 172.16.142.2 (172.16.142.2) 0.015 ms 0.006 ms 0.005 ms
 3 192.168.0.194 (192.168.0.194) 0.014 ms 0.007 ms 0.007 ms
root@n6:/tmp/pycore.45035/n6.conf#

```

Figura 2.24: Conectividade após *Supernetting*

R: Após a eliminação das rotas referentes a Galiza e CDN no dispositivo n6 e com a adição das novas rotas *Supernetting*, podemos afirmar que a conectividade é mantida, tal como podemos observar na figura acima.

2.3.2 b

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

```

root@n6:/tmp/pycore.45035/n6.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags        MSS Window  irtt Iface
10.0.0.0         0.0.0.0        255.255.255.252 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       255.255.255.248 UG           0 0        0 eth0
192.168.0.200    10.0.0.1       255.255.255.248 UG           0 0        0 eth0
192.168.0.208    10.0.0.1       255.255.255.248 UG           0 0        0 eth0
192.168.0.216    10.0.0.1       255.255.255.248 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
root@n6:/tmp/pycore.45035/n6.conf# traceroute 192.168.0.226
traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
 1 10.0.0.6 (10.0.0.6) 0.154 ms 0.007 ms 0.004 ms
 2 10.0.0.18 (10.0.0.18) 0.095 ms 0.008 ms 0.005 ms
 3 10.0.0.22 (10.0.0.22) 0.019 ms 0.007 ms 0.007 ms
 4 10.0.0.26 (10.0.0.26) 0.019 ms 0.011 ms 0.024 ms
 5 10.0.0.30 (10.0.0.30) 0.024 ms 0.013 ms 0.012 ms
 6 172.16.143.2 (172.16.143.2) 0.021 ms 0.021 ms 0.012 ms
 7 192.168.0.226 (192.168.0.226) 0.027 ms 0.015 ms 0.014 ms

```

Figura 2.25: Conectividade após *Supernetting*

2.3.3 c

"Comente os aspetos positivos e negativos do uso do *Supernetting*."

R: O *Supernetting* pode ser vantajoso no sentido de controlar e reduzir o tráfego de rede, pode ainda ser útil para resolver o problema da falta de endereços IP, permite ainda o uso mais eficiente de endereços IP, para além de que possibilita a minimização da tabela de roteamento. Por outro lado, não pode cobrir áreas diferentes de rede quando combinado e ainda aumenta o custo da rede geral. A criação de sub-redes requer roteadores internos, *switches*, *hubs*, etc, que são extremamente caros.

3 Conclusão

A realização deste relatório permitiu com que fundamentássemos, melhor, e de forma mais prática, os conteúdos abordados até ao momento na Unidade Curricular 'Redes de Computadores'. O uso de ferramentas como o *Wireshark* e a aprendizagem de novos comandos foram essenciais para a análise de tráfego de conectividade e identificação de problemas de rede, permitindo assim uma entrada digna a esta área de tão grande importância nos dias de hoje.