

Universidade do Minho
Escola de Engenharia

Redes de Computadores

Relatório do Trabalho Prático nº2
Grupo 41

LEI - 2º Ano - 2º Semestre

Realizador por:
A98695 Lucas Oliveira
A89292 Mike Pinto
A96208 Rafael Gomes

Braga,
17 de fevereiro de 2025

Conteúdo

1 Parte I	5
1.1 Pergunta 1	5
1.1.1 Alínea a)	5
1.1.2 Alínea b)	7
1.1.3 Alínea c)	7
1.1.4 Alínea d)	8
1.2 Pergunta 2	9
1.2.1 Alínea a)	9
1.2.2 Alínea b)	10
1.2.3 Alínea c)	10
1.2.4 Alínea d)	10
1.2.5 Alínea e)	10
1.2.6 Alínea f)	11
1.2.7 Alínea g)	11
1.2.8 Alínea h)	12
1.3 Pergunta 3	14
1.3.1 Alínea a)	14
1.3.2 Alínea b)	15
1.3.3 Alínea c)	15
1.3.4 Alínea d)	16
1.3.5 Alínea e)	17
1.3.6 Alínea f)	18
1.3.7 Alínea g)	18
1.3.8 Alínea h)	18
1.3.9 Alínea i)	18
1.3.10 Alínea j)	19
2 Parte II	21
2.1 Pergunta 1	22
2.1.1 Alínea a)	22
2.1.2 Alínea b)	23
2.1.3 Alínea c)	24
2.1.4 Alínea d)	28
2.1.5 Alínea e)	31
2.1.6 Alínea f)	32
2.1.7 Alínea g)	32
2.2 Pergunta 2	33
2.2.1 Alínea a)	33
2.2.2 Alínea b)	36
2.2.3 Alínea c)	37
2.3 Pergunta 3	42

2.3.1	Alínea a)	42
2.3.2	Alínea b)	45
2.3.3	Alínea c)	47

3 Conclusões	48
---------------------	-----------

Listas de Figuras

1.1.1	Topologia <i>Core</i>	5
1.1.2	Execução do comando "traceroute -I" para o endereço de IP de <i>Found</i> no host <i>Lost</i>	6
1.1.3	Registo de tráfego ICMP do host <i>Lost</i>	6
1.1.4	Comando "traceroute -I" para o servidor <i>Lost</i> com 5 pacotes de prova	7
1.2.1	Comando "traceroute -I" para <i>marco.di.uminho.pt</i>	9
1.2.2	Captura tráfego ICMP de destino a <i>marco.uminho.pt</i>	9
1.2.3	Campo <i>Flags</i> do cabeçalho IP	10
1.2.4	Ordenação do tráfego capturado ICMP pelo campo <i>Source</i>	11
1.2.5	Ordenação do tráfego capturado ICMP pelo campo <i>Destination</i>	12
1.2.6	Cabeçalho IP e ICMP	13
1.3.1	Ping para <i>marco.uminho.pt</i> com o tamanho 3541 bytes.	14
1.3.2	Captura de tráfego pelo Wireshark.	14
1.3.3	Informação de cabeçalho IP e ICMP do primeiro fragmento do datagrama IP original	15
1.3.4	Segundo fragmento do datagrama IP original	16
1.3.5	Terceiro fragmento do datagrama IP original	17
1.3.6	Wireshark filtrado com os últimos fragmentos de cada datagrama IP	17
1.3.7	Comando "Ping -M do -s 2000 marco.uminho.pt"	19
1.3.8	Comando "Ping -M do -s 1472 marco.uminho.pt"	19
1.3.9	Comando "Ping -M do -s 1473 marco.uminho.pt"	20
2.0.1	Topologia <i>Core</i> Caso de Estudo.	21
2.1.1	Comando ping para o servidor <i>Finanças</i>	22
2.1.2	Comando ping para o servidor <i>iTunes</i>	22
2.1.3	Comando ping para o servidor <i>Spotify</i>	22
2.1.4	Comando ping para o servidor <i>Youtube</i>	23
2.1.5	Comando ping para o servidor <i>HBO</i>	23
2.1.6	Comando ping para o servidor <i>Netflix</i>	23
2.1.7	Tabela de encaminhamento do dispositivo <i>AfonsoHenriques</i>	23
2.1.8	Tabela de encaminhamento do dispositivo <i>Teresa</i>	24
2.1.9	Comando "traceroute -I" inicial de <i>AfonsoHenriques</i> para <i>Teresa</i>	24
2.1.10	Tabela de Encaminhamento do router <i>n5</i> da rede <i>core</i>	25
2.1.11	Comando "traceroute -I" de <i>AfonsoHenriques</i> para <i>Teresa</i> após solucionar o problema de encaminhamento do router <i>n5</i> da rede <i>core</i>	25
2.1.12	Tabela de Encaminhamento do router <i>n2</i> da rede <i>core</i>	26
2.1.13	Comando "traceroute -I" de <i>AfonsoHenriques</i> para <i>Teresa</i> após solucionar o problema de encaminhamento do router <i>n2</i> da rede <i>core</i>	27
2.1.14	Tabela de Encaminhamento do router <i>n1</i> da rede <i>core</i>	27
2.1.15	Comando "traceroute -I" de <i>AfonsoHenriques</i> para <i>Teresa</i> após solucionar o problema de encaminhamento da rede <i>core</i>	28
2.1.16	Captura de tráfego dos pacotes ICMP no dispositivo <i>Teresa</i>	28

2.1.17 Tabela de encaminhamento do <i>router RAGaliza</i>	29
2.1.18 Captura de tráfego <i>ICMP</i> no dispositivo <i>Teresa</i> , após a resolução de encaminhamento de pacotes.	29
2.1.19 Comando <i>"traceroute -I"</i> do dispositivo <i>AfonsoHenriques</i> para <i>Teresa</i>	30
2.1.20 Comando <i>"traceroute -I"</i> do dispositivo <i>Teresa</i> para <i>AfonsoHenriques</i>	30
2.1.21 Esquema das rotas usadas pelos pacotes com destino a <i>AfonsoHenriques</i> e <i>Teresa</i>	31
2.1.22 Entrada da tabela de encaminhamento do <i>router n3</i> da rede <i>core</i> fornecida no enunciado	31
2.2.1 Tabela de Encaminhamento do <i>host Castelo</i>	33
2.2.2 Tabela de Encaminhamento do <i>host Castelo</i> sem a sua rota <i>default</i>	33
2.2.3 Tabela de Encaminhamento do <i>host Castelo</i> com ligação à rede <i>Galiza</i>	33
2.2.4 Tabela de Encaminhamento do <i>host Castelo</i> com ligação à rede <i>Institucional</i>	34
2.2.5 Tabela de Encaminhamento do <i>host Castelo</i> final.	34
2.2.6 <i>Ping</i> do dispositivo <i>Castelo</i> para os dispositivos do polo <i>Institucional</i>	35
2.2.7 <i>Ping</i> do dispositivo <i>Castelo</i> para os dispositivos do Polo <i>Galiza</i> e <i>CDN</i>	36
2.2.8 Ligação de um novo <i>host</i> ao <i>router ReiDaNet</i>	38
2.2.9 Comando <i>"ping"</i> do <i>host n33</i> para dispositivos dos Polos <i>CDN</i> e <i>Galiza</i>	39
2.2.10 Comando <i>"ping"</i> do <i>host n33</i> para dispositivos dos Polos <i>Instuticional</i> e <i>Condado Portucalense</i>	40
2.2.11 Comando <i>"ping"</i> do <i>host n33</i> para o dispositivo <i>Castelo</i> do Polo <i>Condado Portucalense</i>	41
2.2.12 Captura de tráfego <i>ICMP</i> do dispositivo <i>Castelo</i> através do <i>wireshark</i>	41
2.3.1 Tabela de encaminhamento do dispositivo <i>n6</i>	42
2.3.2 Tabela de encaminhamento do dispositivo <i>n6</i> sem as rotas referentes a <i>Galiza</i> e <i>CDN</i>	43
2.3.3 Tabela de encaminhamento com a utilização do <i>Supernetting</i> para a <i>Galiza</i> e <i>CDN</i>	44
2.3.4 Ligação estabelecida entre os dispositivos <i>AfonsoHenriques</i> e <i>Teresa</i>	44
2.3.5 Ligação estabelecida entre os dispositivos <i>AfonsoHenriques</i> e <i>Spotify</i>	45
2.3.6 Eliminação das rotas referentes ao <i>Condado Portucalense</i> e <i>Institucional</i> no dispositivo <i>n6</i>	45
2.3.7 Tabela de encaminhamento com a utilização do <i>Supernetting</i> para o <i>Condado Portucalense</i> e <i>Institucional</i>	46
2.3.8 Ligação estabelecida entre os dispositivos <i>Teresa</i> e <i>Castelo</i>	46
2.3.9 Ligação estabelecida entre os dispositivos <i>Teresa</i> e <i>UMinho</i>	46

Capítulo 1

Parte I

1.1 Pergunta 1

Prepare uma topologia CORE para verificar o comportamento do traceroute. Na topologia deve existir: um host (pc) cliente designado Lost, cujo router de acesso é RA1; o router RA1 está simultaneamente ligado a dois routers no core da rede RC1 e RC2; estes estão conectados a um router de acesso RA2, que por sua vez, se liga a um host (servidor) designado Found. 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 15 ms. Após ativar a topologia, note que pode não existir conectividade IP imediata entre Lost e Found até que o anúncio de rotas entre routers estabilize.

Apresentação da topologia Core na figura 1.1.1.

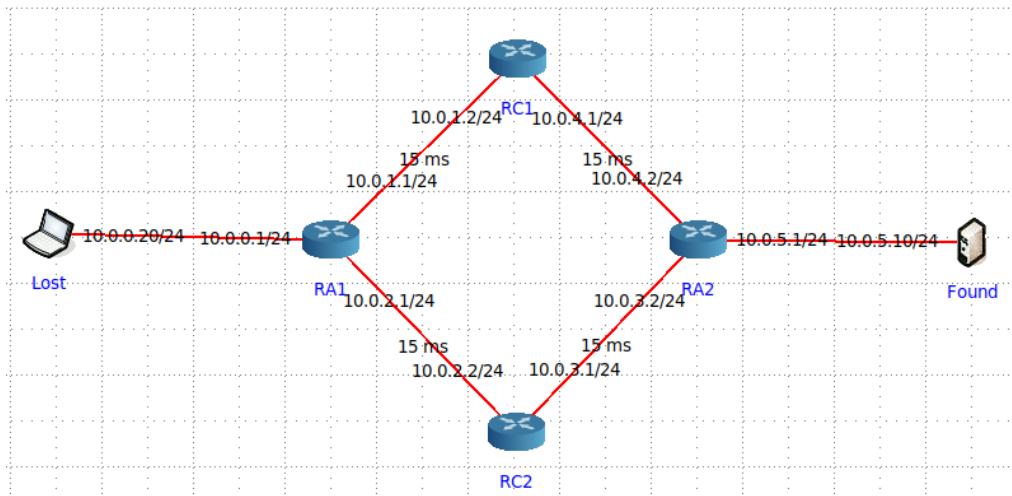


Figura 1.1.1: Topologia Core.

1.1.1 Alínea a)

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.

Ativamos o *Wireshark* no *host Lost* e executamos o comando “*traceroute -I*” para o endereço de *IP* do *Found* como se pode observar na figura 1.1.2

```
root@Lost:/tmp/pycore.39177/Lost.conf# traceroute -I 10.0.5.10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.042 ms  0.009 ms  0.006 ms
 2  10.0.1.2 (10.0.1.2)  31.989 ms  31.979 ms  31.973 ms
 3  10.0.3.2 (10.0.3.2)  63.541 ms  63.539 ms  63.535 ms
 4  10.0.5.10 (10.0.5.10)  63.530 ms  63.526 ms  63.522 ms
```

Figura 1.1.2: Execução do comando ”*traceroute -I*” para o endereço de *IP* de *Found* no *host Lost*.

Após a execução do comando ”*traceroute -I*” e aplicando o filtro no *Wireshark* para protocolo *ICMP* obtemos o seguinte registo de tráfego da figura 1.1.3

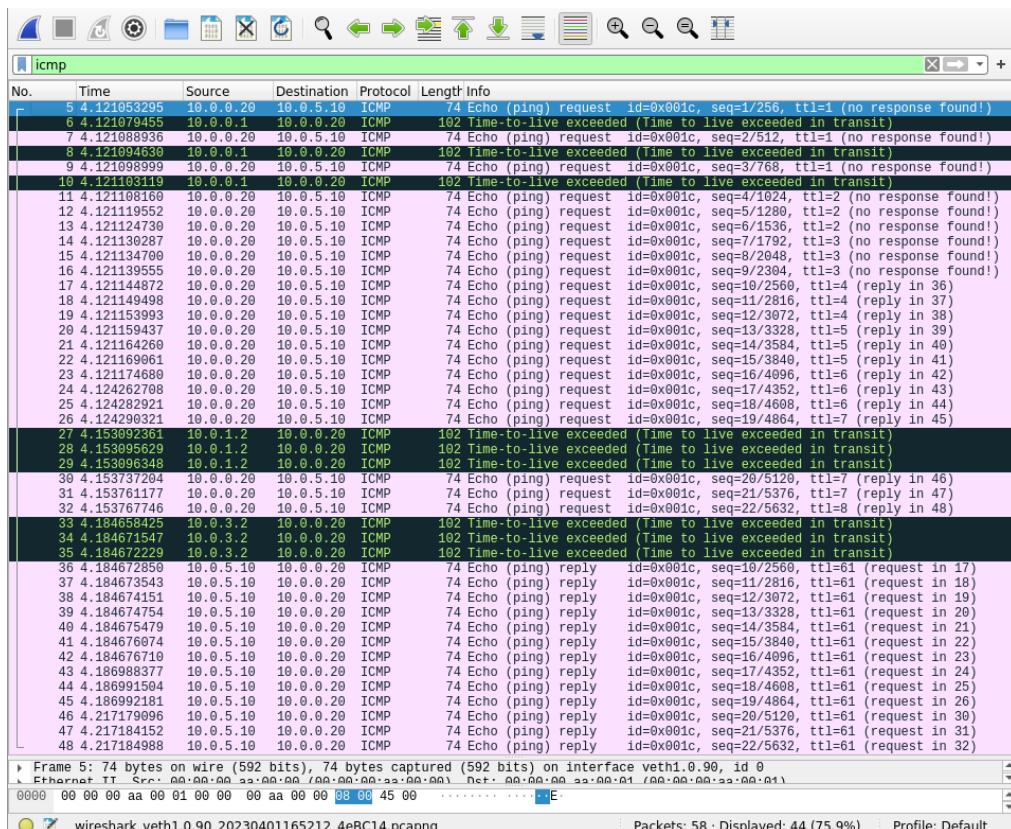


Figura 1.1.3: Registo de tráfego *ICMP* do *host Lost*

Após a análise do registo de tráfego *ICMP* ("Internet Control Message Protocol") do host *Lost* e sabendo que o comando *traceroute* envia vários datagramas *IP* ("Internet Protocol") com valores de *TTL* ("Time-To-Live") incrementaveis, e que cada *router* decrementa este campo em 1, esperamos que enquanto o *TTL* for inferior a 4 o pacote não chegasse ao *Lost*.

É possível verificar que foram inicialmente enviados 3 datagramas com *TTL*=1 que chegaram ao *router RA1*, que decrementou o *TTL*, enviando para o host *Lost* uma mensagem de controlo *ICMP* a indicar que o *TTL* foi excedido. A seguir foi enviado mais três pacotes com o *TTL*=2 que foi decrementado em 1 pelo *router RA1* e *RC1*, onde este ultimo envia uma mensagem de erro, pois o *TTL* dos datagramas atigiram 0. Foram então enviados, novamente, 3 pacotes com o *TTL*=3 que foram decrementados pelos *routers RA1, RC1 e RA2* atingindo neste último o valor de zero, enviando ao host *Lost* outra mensagem de erro.

Por fim, e como era esperado, foram enviados mais 3 pacotes com *TTL*=4 que chegaram ao host *Lost*. Concluímos assim que sempre que o *TTL* < 4 é recebido uma mensagem de erro *ICMP* ("Time-to-live exceeded") como previmos.

1.1.2 Alínea b)

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

O valor inicial mínimo do campo *TTL* para alcançar o servidor *Found* é *TTL*=4, pois todos os pacotes com *TTL* < 4 receberam uma mensagem de erro como podemos observar na figura 1.1.3.

1.1.3 Alínea 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.

Executando o comando "*traceroute -I*" para o servidor *Lost* com 5 pacotes de prova obtermos o resultado da figura 1.1.4

```
root@Lost:/tmp/pycore.45153/Lost.conf# traceroute -I 10.0.5.10 -q 5
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)  0.039 ms  0.007 ms  0.005 ms  0.004 ms  0.005 ms
 2  10.0.1.2 (10.0.1.2)  30.758 ms  30.749 ms  30.744 ms  30.740 ms  30.736 ms
 3  10.0.3.2 (10.0.3.2)  73.145 ms  73.142 ms  73.138 ms  73.135 ms  73.131 ms
 4  10.0.5.10 (10.0.5.10)  73.126 ms  71.547 ms  71.509 ms  71.499 ms  71.490 ms
```

Figura 1.1.4: Comando "*traceroute -I*" para o servidor *Lost* com 5 pacotes de prova

Obtendo assim um *RTT* aproximado de:

$$RTT = (((0.039+0.007+0.005+0.004+0.005)/5)+((30.749+30.749+30.744+30.740+30.736)/5)+(73.145+73.142+73.138+73.135+73.131)/5)+(73.126+71.547+71.509+71.499+71.490)) \times 2 \approx 351,460ms$$

1.1.4 Alínea 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?

Não, o cálculo do valor médio de atraso num sentido (*One-Way Delay*) pode apenas ser *estimado* dividindo o *RTT* por dois, pois teríamos de assumir que as condições de ida e volta do pacote foram idênticas(i.e. mesmo número de saltos, qualidade de transmissão, congestão na rede, entre outras...).

Esta métrica é calculada do ponto **A** até ao ponto **B** recorrendo a relógios sincronizados. Quando o pacote é enviado pelo ponto A e recebido pelo ponto B é registado o “*timestamp*” de ambos os eventos e calculado então o valor médio de atraso num sentido. Esta métrica, numa rede real, torna-se difícil de calcular, por depender muito da sincronização precisa dos relógios.

1.2 Pergunta 2

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)

```
mikep@asus-pc ~ traceroute -I marco.uminho.pt
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 60 byte packets
 1 _gateway (172.26.254.254) 13.061 ms 14.363 ms 14.395 ms
 2 172.16.2.1 (172.16.2.1) 14.320 ms 14.367 ms 14.378 ms
 3 172.16.115.252 (172.16.115.252) 14.384 ms 14.385 ms *
 4 marco.uminho.pt (193.136.9.240) 14.336 ms * *
```

Figura 1.2.1: Comando “traceroute -I” para marco.di.uminho.pt

Foi executado o comando “traceroute -I marco.uminho.pt” numa consola de uma distribuição *linux*, figura 1.2.1, e capturado o tráfego recorrendo ao programa *Wireshark*. Posteriormente foi centrada a análise no primeiro pacote *ICMP* capturado, obtendo o resultado da figura 1.2.2.

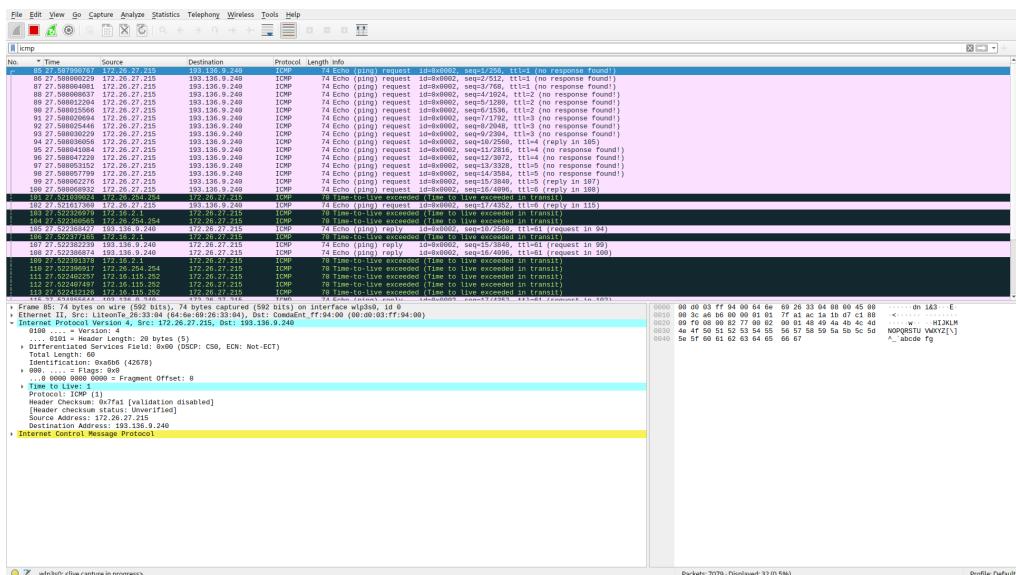


Figura 1.2.2: Captura tráfego ICMP de destino a marco.uminho.pt

1.2.1 Alínea a)

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

Segundo a figura 1.2.2 podemos verificar que a interface ativa do computador é 172.26.27.215 (campo *source* do cabeçalho *IP*).

1.2.2 Alínea b)

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

No campo *protocol* na figura 1.2.2, temos o valor de 1, que identifica o protocolo *ICMP*.

1.2.3 Alínea c)

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

Segundo a figura 1.2.2 o cabeçalho *IP* tem o tamanho de 20 *bytes* (campo *Header Length*), o pacote enviado tem um total de 60 *bytes* (corresponde ao campo *Total Length*).

Para calcular o *payload* basta subtrair o tamanho do cabeçalho *IP* ao tamanho total logo:

$$\text{Payload} = \text{TotalLength} - \text{HeaderLength} = \quad (1.1)$$

$$= 60 - 20 = \quad (1.2)$$

$$= 40\text{bytes} \quad (1.3)$$

Logo o *payload* tem 40 *bytes*.

1.2.4 Alínea d)

O datagrama IP foi fragmentado? Justifique

↙ 000. = Flags: 0x0
 0.... = Reserved bit: Not set
 .0.. = Don't fragment: Not set
 ..0. = More fragments: Not set
 ...0 0000 0000 0000 = Fragment Offset: 0

Figura 1.2.3: Campo *Flags* do cabeçalho *IP*

Não, como podemos verificar na figura 1.2.2 e 1.2.3, o campo *fragment offset* do campo *Flags* tem valor 0, indicando que se existir fragmentação do pacote, este será o primeiro fragmento. Posteriormente, se analisarmos a *flag*, *more fragments* verificamos que o seu valor é 0, logo não existem mais fragmentos. Concluindo assim que o datagrama *IP* não foi fragmentado.

1.2.5 Alínea 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.

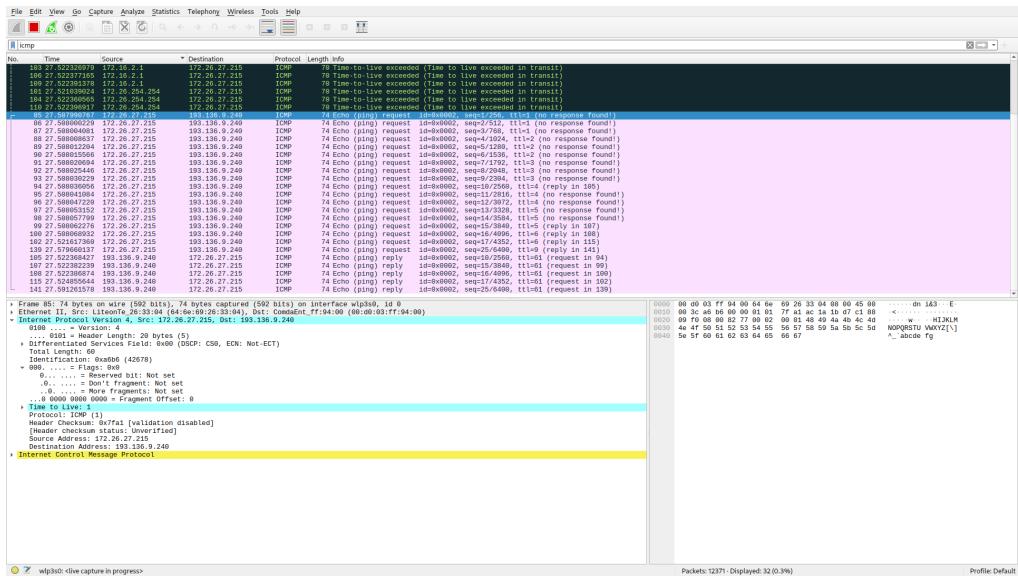


Figura 1.2.4: Ordenação do tráfego capturado *ICMP* pelo campo *Source*

Após a ordenação do tráfego capturado *ICMP* capturado pelo *Wireshark*, figura 1.2.4, é possível verificar que os únicos campos que se alteram de pacote para pacote são os campos *Identification* e *TTL*(*Time-To-Live*).

1.2.6 Alínea f)

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

Sim, enquanto o campo *Identification* incrementa 1 a cada pacote capturado, já o campo *TTL* é incrementado a cada 3 pacotes(i.e. são enviados três pacotes com *TTL*=1, depois são enviados mais três pacotes com *TTL*=2, e assim sucessivamente).

1.2.7 Alínea g)

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

Ordenando o tráfego capturado por endereço de destino obtemos o resultado da figura 1.2.5

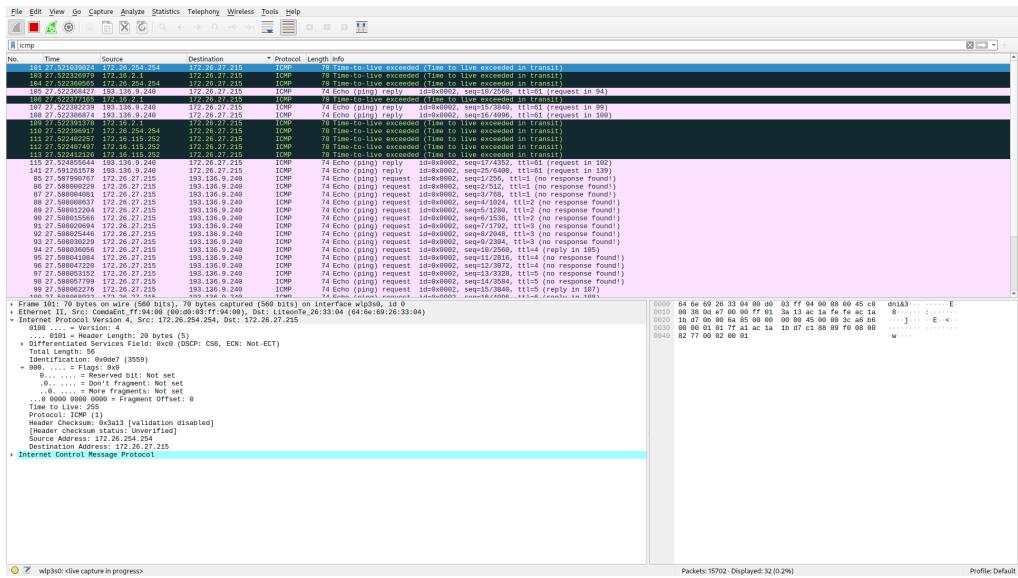


Figura 1.2.5: Ordenação do tráfego capturado *ICMP* pelo campo *Destination*

Subalínea 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?

Foram recebidas 9 mensagens de resposta *ICMP TTL Exceeded* com diferentes valores de *TTL*, três com o valor de *TTL*=255, outras três com *TTL*=254 e outras três com *TTL*=253.

Os valores de *TTL*=255 são a resposta às mensagens *ICMP* com o *TTL*=1, as mensagens com *TTL*=254 são a resposta às mensagens *ICMP* com o *TTL*=2 e as mensagens com *TTL*=253 são a resposta às mensagens *ICMP* com o *TTL*=3.

O significado dos diferentes valores de *TTL* recebidos, é devido à resposta ter vindo de diferentes *routers* no caminho entre a origem e o destino.

Subalínea ii)

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

As mensagens de resposta *ICMP TTL Exceeded* são enviadas com um valor de *TTL* relativamente alto para se ter a certeza que a mensagem é recebida por quem enviou o pacote *ICMP*.

1.2.8 Alínea 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?

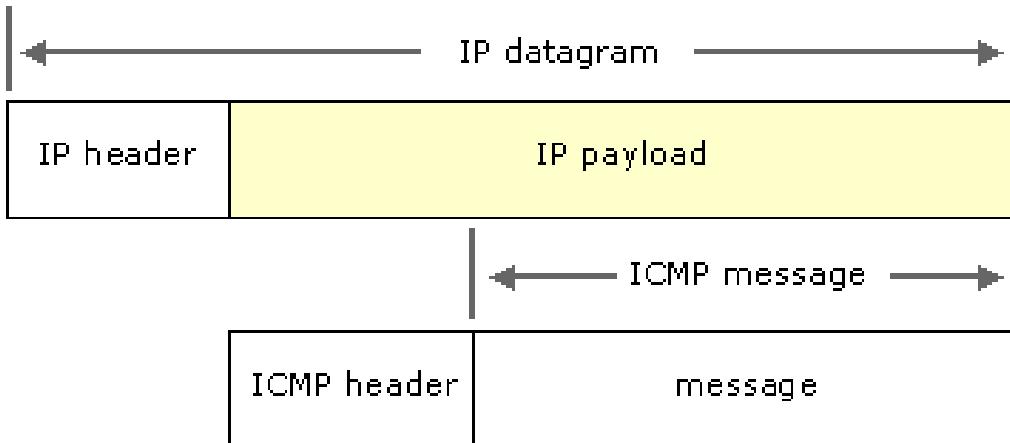


Figura 1.2.6: Cabeçalho *IP* e *ICMP*

Sim, a informação do cabeçalho *ICMP* poderia ser hipoteticamente incluída no cabeçalho *IPv4*. A principal vantagem desta inclusão seria que o remetente de um pacote saberia sempre se este foi entregue com sucesso ou se ocorreu algum erro na sua transmissão.

Por outro lado, essa inclusão iria levar a um aumento desnecessário de tamanho do cabeçalho *IPv4*, figura 1.2.6, sempre que um pacote chegasse ao destino com sucesso, para além de poder trazer problemas de fragmentação e desempenho de rede, sendo necessário mais tempo de transmissão dos pacotes.

1.3 Pergunta 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 ($3500 + X$) bytes, em que X é o número do grupo de trabalho (e.g., $X=22$ para o grupo PL22).

Realizamos então o *ping* para *marco.uminho.pt* com o tamanho de 3541 bytes devido a sermos o grupo *PL41*, figura 1.3.1.

```
mikep@asus-pc ~ ping -s 3541 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 3541(3569) bytes of data.
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=1 ttl=61 time=15.9 ms
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=2 ttl=61 time=4.99 ms
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=3 ttl=61 time=44.0 ms
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=4 ttl=61 time=3.22 ms
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=5 ttl=61 time=29.8 ms
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=6 ttl=61 time=3.45 ms
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=7 ttl=61 time=5.91 ms
3549 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=8 ttl=61 time=57.5 ms
^C
--- marco.uminho.pt ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7011ms
rtt min/avg/max/mdev = 3.219/20.596/57.484/19.600 ms
```

Figura 1.3.1: *Ping* para *marco.uminho.pt* com o tamanho 3541 bytes.

Capturamos o tráfego com recurso ao *Wireshark*, como demonstrado na figura 1.3.2.

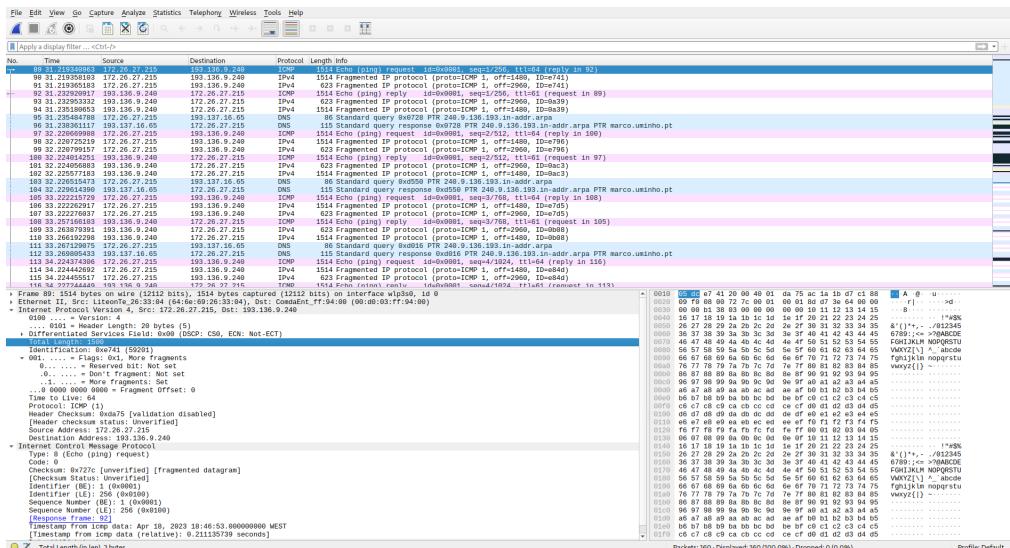


Figura 1.3.2: Captura de tráfego pelo *Wireshark*.

1.3.1 Alínea a)

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

Como podemos observar na figura 1.3.2, o tamanho máximo do pacote é de 1500 *bytes*, e como enviamos pacotes com o tamanho de 3541 *bytes*, foi necessário fragmentar o pacote para poder circular na rede.

1.3.2 Alínea 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?

```
- Internet Protocol Version 4, Src: 172.26.27.215, 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: 0xe741 (59201)
  ▶ 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: 0xda75 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.27.215
  Destination Address: 193.136.9.240
  ▶ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x727c [unverified] [fragmented datagram]
    [Checksum Status: Unverified]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence Number (BE): 1 (0x0001)
    Sequence Number (LE): 256 (0x0100)
    [Response frame: 92]
    Timestamp from icmp data: Apr 18, 2023 18:46:53.000000000 WEST
    [Timestamp from icmp data (relative): 0.211135739 seconds]
```

Figura 1.3.3: Informação de cabeçalho *IP* e *ICMP* do primeiro fragmento do datagrama *IP* original

Como podemos observar na figura 1.3.3 no campo *flags* do cabeçalho verificamos que existem mais fragmentos(*more fragments*), logo o datagrama foi fragmentado. Trata-se do primeiro fragmento, porque o *fragment offset* está definido a 0, o que indica que é o primeiro fragmento. O total *length* é de 1500 *bytes*, sabemos ainda que o cabeçalho possui 20 *bytes* de tamanho e que o cabeçalho *ICMP* ocupa 8 *bytes* tendo então 1472 *bytes* de dados.

1.3.3 Alínea 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?

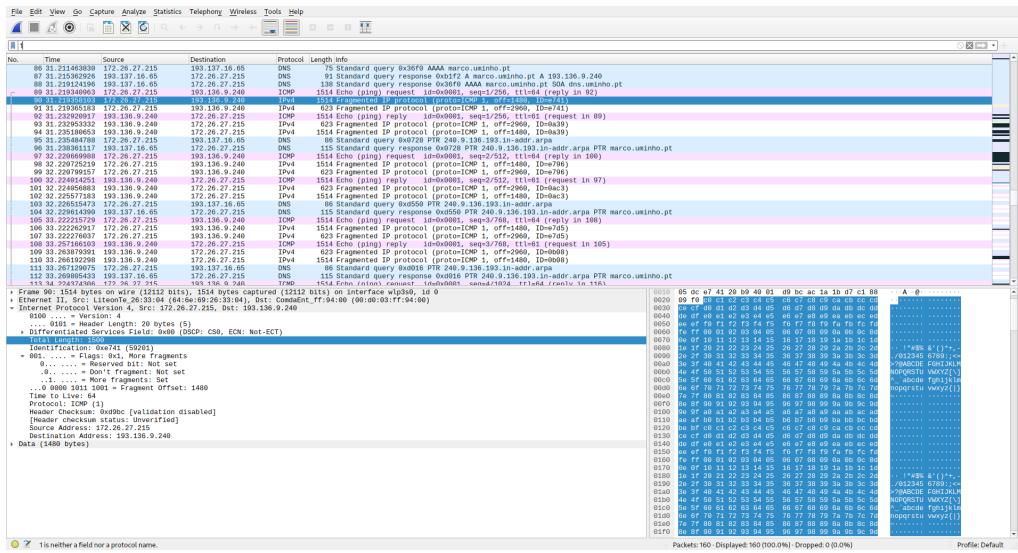


Figura 1.3.4: Segundo fragmento do datagrama *IP* original

Como podemos observar na figura 1.3.4 o *fragment offset* é diferente de 0 (tem o valor de 1480) o que nos indica que este é o segundo fragmento do datagrama. Existem mais fragmentos devido no campo *flags* o campo *more fragments* ter o valor *set*.

1.3.4 Alínea 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.

O tamanho do pacote enviado foi de 3541 *bytes*, sabendo que no máximo cada pacote possui 1500 *bytes*, então precisamos de pelo menos 3 fragmentos para enviar o pacote completo.

Sabendo que o cabeçalho *IPv4* possui um *header* de 20 *bytes* e que o tamanho máximo de um pacote é de 1500 *bytes*, então temos no máximo 1480 *bytes* de data, sendo que no primeiro pacote ainda temos o cabeçalho *ICMP* que ocupa 8 *bytes* ficando no máximo, 1472 *bytes* de dados para o primeiro fragmento. O tamanho de *bytes* enviados foi de 3541, logo $3541 - 1472 - 1480 = 589$ *bytes* (Devido a sabermos que foi fragmentado pelo menos duas vezes das alíneas anteriores).

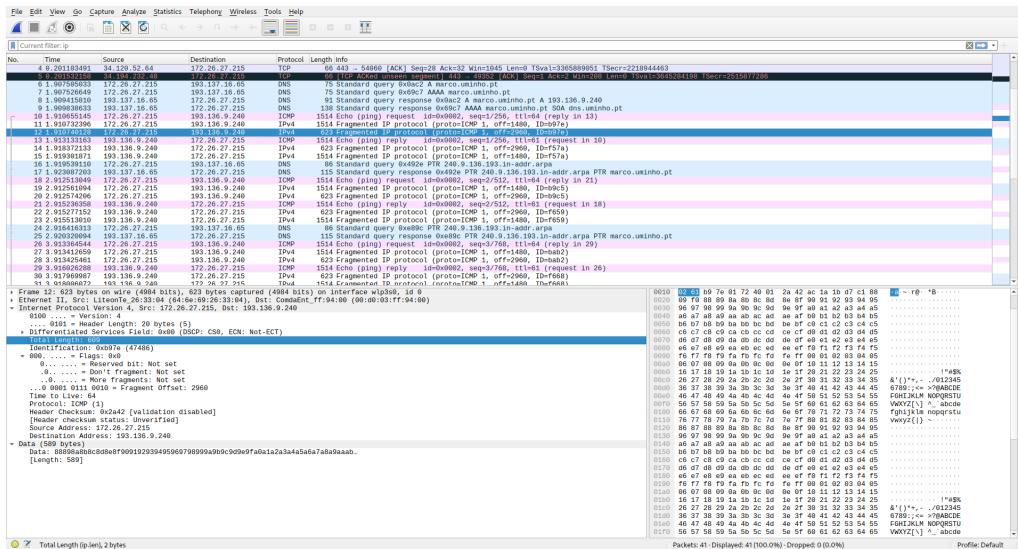


Figura 1.3.5: Terceiro fragmento do datagrama IP original

Após a análise da captura de tráfego do *wireshark* observamos que de facto, o pacote foi fragmentado pelo menos três vezes. O número de *bytes* do último fragmento IP foi de 609, retirando o tamanho do cabeçalho obtemos então $609 - 20 = 589$ bytes, correspondendo ao valor estimado.

1.3.5 Alínea 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

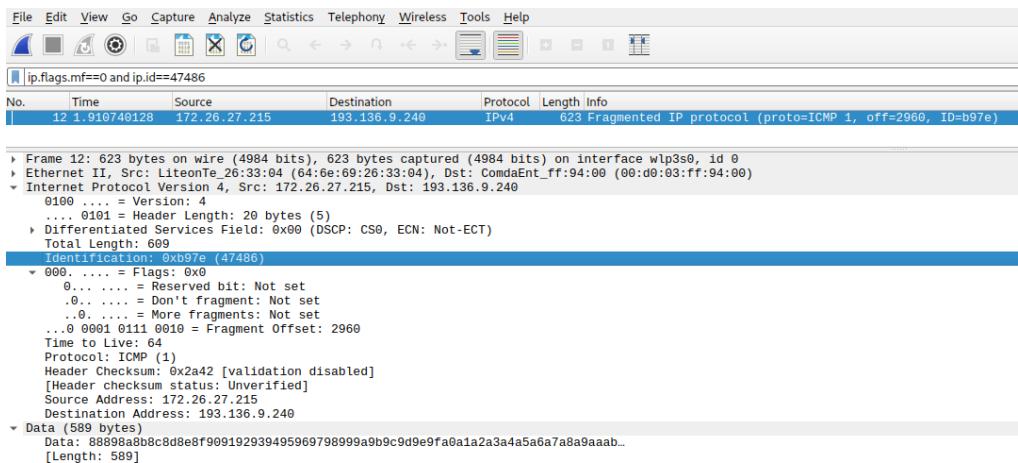


Figura 1.3.6: *Wireshark* filtrado com os últimos fragmentos de cada datagrama IP

Para detetar o último fragmento de um datagrama é necessário comparar se o campo *identification* do cabeçalho *IPv4* possui o mesmo valor do primeiro fragmento encontrado e se o campo *flags* possui o campo *more fragments* com o valor de *not set*. Para listarmos

os últimos fragmentos dos primeiros datagramas *IPs* segmentados utilizamos o filtro "*ip.flags.mf=0 and ip.id=-47486*" como podemos observar na figura 1.3.6.

1.3.6 Alínea 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?

O datagrama *IP* original é reconstruído no dispositivo */host* de destino, não podendo ser reconstruído noutro equipamento devido à capacidade máxima da rede. Se o datagrama fosse reconstruído a meio da sua "viagem" desde a origem ao destino, este não poderia ser enviado novamente completo devido ao seu tamanho ser superior à capacidade máxima da rede, sendo necessário fragmentar novamente o datagrama.

1.3.7 Alínea 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.

Os campos que mudam no cabeçalho *IP* entre os diferentes fragmentos são os campos, *fragment Offset* e o campo *More fragments*.

Estes campos permitem reconstruir o datagrama original, pois o campo *fragment Offset* indica qual a posição do fragmento no datagrama original e o campo *more fragments* indica se existem mais fragmentos ou não daquele datagrama.

1.3.8 Alínea h)

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

Sabemos que o protocolo *ICMP* é utilizado para fornecer relatórios de erros do envio de um pacote. A principal razão do primeiro fragmento de cada pacote ser identificado como pacote *ICMP* é informar se existe algum problema de ligação entre a origem e o destino.

1.3.9 Alínea 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?

O valor utilizado para comparar o tamanho de um datagrama de modo a se determinar se este deve ser fragmentado é o valor do *MTU*(*Maximum Transmission Unit*), por norma 1500 bytes.

O aumento do valor do *MTU* levaria a uma menor sobrecarga na rede, devido à circular uma menor quantidade de pacotes, mas por consequência, estes pacotes seriam maiores, resultando numa diminuição na velocidade de transmissão e no aumento de erros na sua transmissão (nomeadamente perdas de pacotes).

Em contraste, com a diminuição do valor de *MTU* levaria a uma maior sobrecarga na rede, devido a existirem uma maior quantidade de pacotes a circular, mas por consequência teríamos um aumento na fragmentação dos pacotes, levando a um desgaste

desnecessário na inclusão de novos cabeçalhos aos pacotes fragmentados.

Quando um pacote é fragmentado e se perde na sua transmissão é possível pedir a retransmissão apenas desse fragmento, por outro lado, quando um pacote que não foi fragmentado se perde na sua transmissão é necessário pedir a sua retransmissão, levando a uma maior sobrecarga na rede.

1.3.10 Alínea 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.

Executamos então o comando *“ping -M do -s 2000 marco.uminho.pt”* de forma a descobrir qual o valor do *MTU*(*Maximum Transmission Unit*) obtendo o resultado da figura 1.3.7

```
mikep@asus-pc ~ > ping -M do -s 2000 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 2000(2028) 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
^C
--- marco.uminho.pt ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2025ms
```

Figura 1.3.7: Comando *“Ping -M do -s 2000 marco.uminho.pt”*

Verificamos assim que o *MTU* = 1500. Sabendo que o cabeçalho *IP* tem o tamanho de 20 *bytes* e que o tamanho do cabeçalho *ICMP* é de 8 *bytes* temos então: $1500 - 20 - 8 = 1472\ bytes$. Executando então o comando *“ping -M do -s 1472 marco.uminho.pt”* verificamos que o pacote foi enviado, figura 1.3.8.

Para confirmar que 1472 *bytes* era o valor máximo sem ocorrer fragmentação executamos o comando anterior, mas desta vez com o tamanho de 1473 *bytes* (*“ping -M do -s 1473 marco.uminho.pt”*) obtendo uma mensagem de erro, figura 1.3.9, confirmando assim a nossa dedução de 1472 *bytes*.

```
x mikep@asus-pc ~ > 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=38.5 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=2 ttl=61 time=24.6 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=3 ttl=61 time=65.5 ms
^C
--- marco.uminho.pt ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 24.615/42.863/65.481/16.967 ms
```

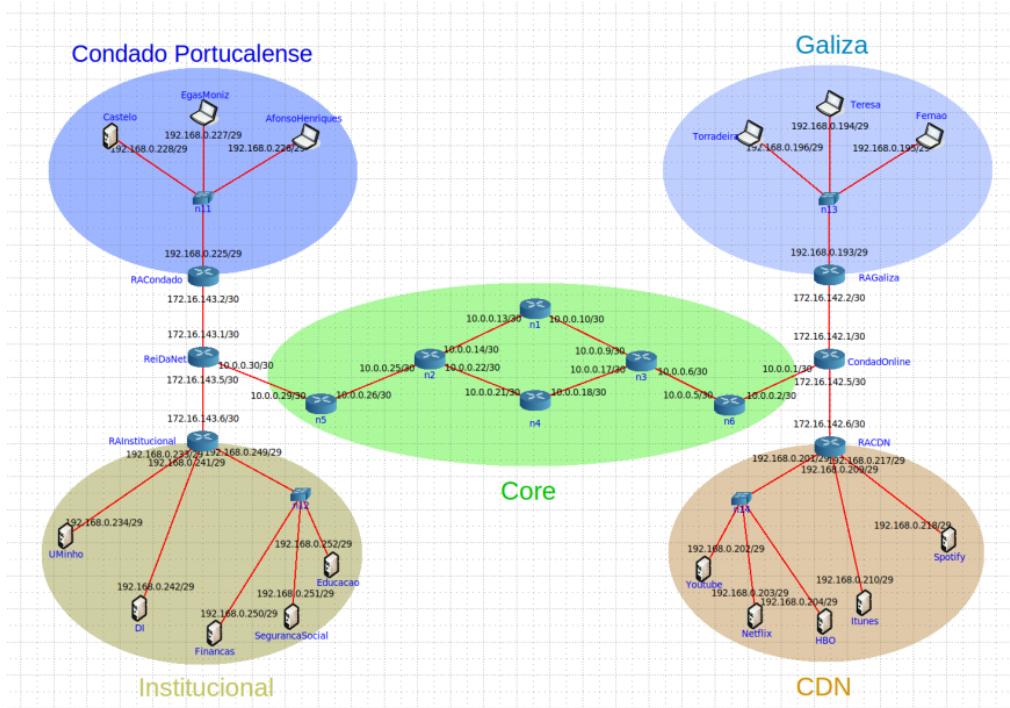
Figura 1.3.8: Comando *“Ping -M do -s 1472 marco.uminho.pt”*

```
mikep@asus-pc ~ 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
^C
--- marco.uminho.pt ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2023ms
```

Figura 1.3.9: Comando "*Ping -M do -s 1473 marco.uminho.pt*"

Capítulo 2

Parte II



2.1 Pergunta 1

D.Afonso Henriques afirma ter problemas de comunicação com a sua mãe, D.Teresa. 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.

2.1.1 Alínea a)

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

Após a execução do comando "*ping*" para o servidor das *Finanças*, figura 2.1.1, e os vários servidores de *CDN*, figuras 2.1.2, 2.1.3, 2.1.4, 2.1.5 e 2.1.6, averiguamos que *AfonsoHenriques* tem efetivamente conectividade com o servidor *Finanças* e os servidores da *CDN*.

```
<core.35467/AfonsoHenriques.conf# ping 192.168.0.242
PING 192.168.0.242 (192.168.0.242) 56(84) bytes of data.
64 bytes from 192.168.0.242: icmp_seq=1 ttl=61 time=0.156 ms
64 bytes from 192.168.0.242: icmp_seq=2 ttl=61 time=0.175 ms
64 bytes from 192.168.0.242: icmp_seq=3 ttl=61 time=0.066 ms
64 bytes from 192.168.0.242: icmp_seq=4 ttl=61 time=0.072 ms
64 bytes from 192.168.0.242: icmp_seq=5 ttl=61 time=0.194 ms
64 bytes from 192.168.0.242: icmp_seq=6 ttl=61 time=0.162 ms
64 bytes from 192.168.0.242: icmp_seq=7 ttl=61 time=0.132 ms
64 bytes from 192.168.0.242: icmp_seq=8 ttl=61 time=0.192 ms
```

Figura 2.1.1: Comando *ping* para o servidor *Finanças*.

```
<core.35467/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=0.155 ms
64 bytes from 192.168.0.210: icmp_seq=2 ttl=55 time=0.154 ms
64 bytes from 192.168.0.210: icmp_seq=3 ttl=55 time=0.158 ms
```

Figura 2.1.2: Comando *ping* para o servidor *iTunes*.

```
<core.35467/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=0.158 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=55 time=0.111 ms
64 bytes from 192.168.0.218: icmp_seq=3 ttl=55 time=0.225 ms
```

Figura 2.1.3: Comando *ping* para o servidor *Spotify*.

```
<core.35467/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=0.158 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.195 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.309 ms
```

Figura 2.1.4: Comando *ping* para o servidor *Youtube*.

```
<core.35467/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=0.152 ms
64 bytes from 192.168.0.204: icmp_seq=2 ttl=55 time=0.555 ms
64 bytes from 192.168.0.204: icmp_seq=3 ttl=55 time=0.112 ms
64 bytes from 192.168.0.204: icmp_seq=4 ttl=55 time=0.314 ms
```

Figura 2.1.5: Comando *ping* para o servidor *HBO*.

```
<core.35467/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=0.152 ms
64 bytes from 192.168.0.203: icmp_seq=2 ttl=55 time=0.555 ms
64 bytes from 192.168.0.203: icmp_seq=3 ttl=55 time=0.311 ms
64 bytes from 192.168.0.203: icmp_seq=4 ttl=55 time=0.175 ms
```

Figura 2.1.6: Comando *ping* para o servidor *Netflix*.

2.1.2 Alínea 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.

Sabendo que a coluna "*Destination*", de uma tabela de encaminhamento, indica a sub-rede de destino, a coluna "*Gateway*" indica qual o próximo equipamento que o pacote irá ser enviado e a coluna "*Genmask*" indica o tipo de máscara que será usada então, através da execução do comando "*netstat -rn*" nos dispositivos de *AfonsoHenriques* e *Teresa* obtemos as tabelas de encaminhamento das figuras 2.1.7 e 2.1.8 respectivamente.

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 2.1.7: Tabela de encaminhamento do dispositivo *AfonsoHenriques*.

```

root@Teresa:/tmp/pycore.35467/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

```

Figura 2.1.8: Tabela de encaminhamento do dispositivo *Teresa*.

Após análise de ambas as tabelas é possível averiguar que estas não possuem nenhum problema. Em ambos os casos só existem duas entradas, quando o pacote tiver o endereço de destino da sub-rede do dispositivo, no caso de *AfonsoHenriques* o endereço da sub-rede do *Condado Portucalense* e no caso de *Teresa* o endereço da sub-rede *Galiza*, então pode optar por qualquer destino da sub-rede, por outro lado, quando o endereço de destino do pacote é diferente do endereço da sub-rede do dispositivo então o pacote irá ser encaminhado, no caso de *AfonsoHenriques*, para o *router RACondade*, e no caso de *Teresa* para o *router RAGaliza*.

2.1.3 Alínea c)

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*.

Por sugestão do docente do nosso turno prático, foi utilizado o comando `"traceroute -I"` em vez do *WireShark*.

Usando este procedimento foram encontrados problemas de encaminhamento de pacotes nos dispositivos *n5*, *n2* e *n1* da rede *core*.

Inicialmente, executamos o comando `"traceroute -I 192.168.0.194"` no dispositivo *AfonsoHenriques* obtendo o seguinte resultado da figura 2.1.9 verificando que o pacote chegava somente ao *router n5* da rede *core*.

```

<pycore.35467/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)  0.085 ms  0.008 ms  0.005 ms
2  172.16.143.1 (172.16.143.1)  0.021 ms  0.007 ms  0.007 ms
3  10.0.0.29 (10.0.0.29)  0.051 ms !N  0.011 ms !N *

```

Figura 2.1.9: Comando `"traceroute -I"` inicial de *AfonsoHenriques* para *Teresa*.

Executamos então o comando `"netstat -rn"` no *router n5* da rede *core*, onde verificamos que não existia nenhuma entrada para o encaminhamento de pacotes para a sub-rede *Galiza*, como podemos verificar na figura 2.1.10.

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

Figura 2.1.10: Tabela de Encaminhamento do *router n5* da rede *core*.

Então executamos o comando *"route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.25"*, adicionando uma nova entrada na tabela de encaminhamento especificando que todos os pacotes com o destino para a subrede *Galiza* teriam de ser enviados para o *router n2* da rede *core*.

Após executar o comando *"traceroute -I 192.168.0.194"* novamente no dispositivo de *AfonsoHenriques* obtemos o resultado da figura 2.1.11, onde após analise verificamos que o *router n2* enviava o pacote novamente para o *router n5*.

```
<Pycore.35467/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)  0.037 ms  0.007 ms  0.005 ms
 2  172.16.143.1 (172.16.143.1)  0.018 ms  0.009 ms  0.008 ms
 3  10.0.0.29 (10.0.0.29)  0.021 ms  0.011 ms  0.012 ms
 4  10.0.0.25 (10.0.0.25)  0.022 ms  0.015 ms  0.013 ms
 5  10.0.0.25 (10.0.0.25)  3068.704 ms !H  3068.686 ms !H  3068.679 ms !H
```

Figura 2.1.11: Comando *"traceroute -I"* de *AfonsoHenriques* para *Teresa* após solucionar o problema de encaminhamento do *router n5* da rede *core*.

Executando o comando *"netstat -rn"* no *router n2* da rede *core* obtemos a tabela de encaminhamento da figura 2.1.12.

Kernel IP routing table							
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
192.168.0.194	10.0.0.25	255.255.255.254	UG	0	0	0	eth2
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 2.1.12: Tabela de Encaminhamento do *router n2* da rede *core*.

Confirmando assim que o *router n2* envia todos os pacotes com destino ao endereço *IP* de *Teresa* para o *router n5*. Foi então executado o comando ”*route del -net 192.168.0.194 netmask 255.255.255.254*” apagando a entrada que dava problemas no encaminhamento. Não foi necessário adicionar nenhuma entrada na tabela, pois já existe uma entrada para encaminhamento de pacotes com destino à rede *Galiza*.

Foi executado novamente o comando ”*traceroute -I*” no dispositivo *AfonsoHenriques* onde verificamos que o *router n1* envia o pacote novamente para o *router n2*, estando o pacote em ”*loop*”, como é possível verificar na figura 2.1.13

```

Kycore.35467/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)  0.038 ms  0.006 ms  0.005 ms
 2  172.16.143.1 (172.16.143.1)  0.018 ms  0.007 ms  0.007 ms
 3  10.0.0.29 (10.0.0.29)  0.019 ms  0.011 ms  0.009 ms
 4  10.0.0.25 (10.0.0.25)  0.022 ms  0.012 ms  0.012 ms
 5  10.0.0.13 (10.0.0.13)  0.025 ms  0.015 ms  0.014 ms
 6  10.0.0.25 (10.0.0.25)  0.014 ms  0.221 ms  0.030 ms
 7  10.0.0.13 (10.0.0.13)  0.018 ms  0.016 ms  0.016 ms
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * 10.0.0.13 (10.0.0.13)  0.300 ms  0.088 ms
14  10.0.0.25 (10.0.0.25)  0.082 ms  0.075 ms  0.079 ms
15  10.0.0.13 (10.0.0.13)  0.079 ms  0.083 ms  0.081 ms
16  10.0.0.25 (10.0.0.25)  0.081 ms  0.081 ms *^C

```

Figura 2.1.13: Comando *"traceroute -I"* de AfonsoHenriques para *Teresa* após solucionar o problema de encaminhamento do *router n2* da rede *core*.

Executando o comando *"netstat -rn"* no *router n1* obtemos a seguinte tabela de encaminhamento da figura 2.1.14

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

Figura 2.1.14: Tabela de Encaminhamento do *router n1* da rede *core*.

Neste caso foram executados dois comandos:

- **Primeiro:** *"route del -net 192.168.0.192 netmask 255.255.255.248"*.
- **Segundo:** *"route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.9"*

Após corrigido as entradas erradas das tabelas de encaminhamento, ao executar o comando *"traceroute -I 192.168.0.194"* verificamos que os pacotes chegam ao *router CondadOnline* como demonstrado na figura 2.1.15

```
Kycore.35467/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)  0.084 ms  0.008 ms  0.006 ms
 2  172.16.143.1 (172.16.143.1)  0.025 ms  0.009 ms  0.007 ms
 3  10.0.0.29 (10.0.0.29)  0.040 ms  0.011 ms  0.011 ms
 4  10.0.0.25 (10.0.0.25)  0.039 ms  0.013 ms  0.013 ms
 5  10.0.0.13 (10.0.0.13)  0.043 ms  0.016 ms  0.015 ms
 6  10.0.0.17 (10.0.0.17)  0.079 ms  0.073 ms  0.020 ms
 7  10.0.0.5 (10.0.0.5)  0.059 ms  0.022 ms  0.021 ms
 8  10.0.0.1 (10.0.0.1)  0.052 ms  0.024 ms  0.024 ms
```

Figura 2.1.15: Comando *"traceroute -I"* de *AfonsoHenriques* para *Teresa* após solucionar o problema de encaminhamento da rede *core*.

2.1.4 Alínea 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*.

Executamos então o *Wireshark* no dispositivo *Teresa* e o comando *"traceroute -I 192.168.0.194"* no dispositivo *AfonsoHenriques*. Após analisar a captura de tráfego *ICMP* do dispositivo *Teresa* é possível verificar que os pacotes enviados por *AfonsoHenriques* são recebidos em *Teresa*, como demonstrado na figura 2.1.16

19	28	756864668	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b seq=1/256 ttl=1 (no response found!)
20	28	756863344	192.168.0.194	192.168.0.194	ICMP	102 Destination unreachable (Network unreachable)
21	28	756397293	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=512, ttl=1 (no response found!)
22	28	756405478	192.168.0.194	192.168.0.194	ICMP	102 Destination unreachable (Network unreachable)
23	28	756414986	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response found!)
24	28	756424163	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no response found!)
25	28	756431653	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no response found!)
26	28	756439108	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no response found!)
27	28	756447656	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no response found!)
28	28	756447656	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no response found!)
29	28	756462307	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no response found!)
30	28	756471124	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=10/2560, ttl=3 (no response found!)
31	28	756479818	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=11/2816, ttl=3 (no response found!)
32	28	756486979	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (no response found!)
33	28	756494612	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (no response found!)
34	28	756502569	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (no response found!)
35	28	756510377	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (no response found!)
36	28	756518970	192.168.0.194	192.168.0.226	ICMP	74 Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (no response found!)

Figura 2.1.16: Captura de tráfego dos pacotes *ICMP* no dispositivo *Teresa*.

Subalínea 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.

Após análise da captura de tráfego de pacotes *ICMP* no dispositivo *Teresa*, verificamos que o *RAGaliza* envia uma mensagem de erro *"Destination Unreachable"*.

Executando o comando *"netstat -rn"* no router *RAGaliza* obtemos a tabela de encaminhamento da figura 2.1.17.

root@RAGaliza:/tmp/pycore.44765/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
192.168.0.232	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

Figura 2.1.17: Tabela de encaminhamento do *router RAGaliza*.

Após a análise da tabela de encaminhamento, verificamos não existir nenhuma entrada de encaminhamento de pacotes para a rede *Condado Portucalense*. Executamos então o comando *"route add -net 192.168.0.224 netmask 255.255.255.248 gw 172.16.142.1"*.

Repetimos o processo de captura de tráfego de pacotes ICMP com o *Wireshark* no dispositivo de *Teresa* onde podemos observar que já existe conexão entre os dois dispositivos, figura 2.1.18.

19	29	2450707216	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) request	id=0x0037, seq=28/7160, ttl=1 (reply in 28)
20	29	2450707216	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) reply	id=0x0037, seq=28/7160, ttl=64 (request in 19)
21	29	2450623908	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) request	id=0x0037, seq=29/7424, ttl=1 (reply in 20)
22	29	2450623908	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) reply	id=0x0037, seq=29/7424, ttl=64 (request in 21)
23	29	2460326885	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) request	id=0x0037, seq=30/7680, ttl=1 (reply in 24)
24	29	2460379985	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) reply	id=0x0037, seq=30/7680, ttl=64 (request in 23)
25	29	246103545	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) request	id=0x0037, seq=31/7936, ttl=2 (reply in 26)
26	29	246108771	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) reply	id=0x0037, seq=31/7936, ttl=64 (request in 25)
27	29	246173210	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) request	id=0x0037, seq=32/8192, ttl=2 (reply in 28)
28	29	246178390	192.168.0.226	192.168.0.194	ICMP	74	Echo (ping) reply	id=0x0037, seq=32/8192, ttl=64 (request in 27)

Figura 2.1.18: Captura de tráfego ICMP no dispositivo *Teresa*, após a resolução de encaminhamento de pacotes.

Subalínea 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.

Executando o comando *"traceroute -I"* nos dispositivos de *AfonsoHenrique* e *Teresa*, obtemos o resultado das figuras 2.1.19 e 2.1.20

```
<pycore.44765/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)  0.032 ms  0.006 ms  0.004 ms
 2  172.16.143.1 (172.16.143.1)  0.017 ms  0.048 ms  0.010 ms
 3  10.0.0.29 (10.0.0.29)  0.018 ms  0.009 ms  0.009 ms
 4  10.0.0.25 (10.0.0.25)  0.082 ms  0.014 ms  0.011 ms
 5  10.0.0.13 (10.0.0.13)  0.022 ms  0.013 ms  0.013 ms
 6  10.0.0.17 (10.0.0.17)  0.027 ms  0.065 ms  0.028 ms
 7  10.0.0.5 (10.0.0.5)  0.041 ms  0.021 ms  0.019 ms
 8  10.0.0.1 (10.0.0.1)  0.030 ms  0.023 ms  0.021 ms
 9  172.16.142.2 (172.16.142.2)  0.035 ms  0.023 ms  0.041 ms
10  192.168.0.194 (192.168.0.194)  0.058 ms  0.050 ms  0.030 ms
```

Figura 2.1.19: Comando *"traceroute -I"* do dispositivo *AfonsoHenriques* para *Teresa*.

```
root@Teresa:/tmp/pycore.44765/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)  0.055 ms  0.012 ms  0.009 ms
 2  172.16.142.1 (172.16.142.1)  0.029 ms  0.012 ms  0.012 ms
 3  10.0.0.2 (10.0.0.2)  0.030 ms  0.021 ms  0.021 ms
 4  10.0.0.6 (10.0.0.6)  0.036 ms  0.021 ms  0.025 ms
 5  10.0.0.18 (10.0.0.18)  0.035 ms  0.028 ms  0.018 ms
 6  10.0.0.14 (10.0.0.14)  0.036 ms  0.083 ms  0.022 ms
 7  10.0.0.26 (10.0.0.26)  0.031 ms  0.020 ms  0.019 ms
 8  10.0.0.30 (10.0.0.30)  0.034 ms  0.040 ms  0.289 ms
 9  172.16.143.2 (172.16.143.2)  0.126 ms  0.052 ms  0.031 ms
10  192.168.0.226 (192.168.0.226)  0.038 ms  0.039 ms  0.027 ms
root@Teresa:/tmp/pycore.44765/Teresa.conf# []
```

Figura 2.1.20: Comando *"traceroute -I"* do dispositivo *Teresa* para *AfonsoHenriques*.

Após a análise das rotas realizadas pelos pacotes, obtemos as rotas do esquema da figura 2.1.21. Logo, por exemplo, quando realizamos o comando *"traceroute -I"* para o dispositivo *Teresa* a partir de *AfonsoHenriques*, verificamos que a resposta ao *Echo request* no *router n3* tem origem na interface com o endereço *10.0.0.17*, como é possível verificar na figura 2.1.19 e 2.1.21.

Concluímos assim que as rotas entre os *Echo Request* e *Echo Reply* são diferentes entre o dispositivo *AfonsoHenriques* e *Teresa*.

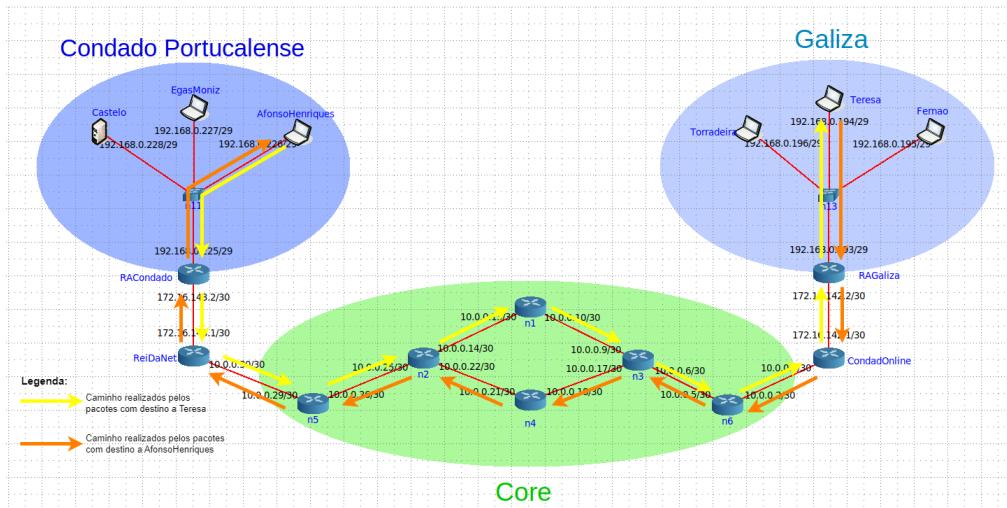


Figura 2.1.21: Esquema das rotas usadas pelos pacotes com destino a *AfonsoHenriques* e *Teresa*.

2.1.5 Alínea e)

Estando restabelecida a conectividade entre os dois hosts, obtenha a tabela de encaminhamento de *n3* e foque-se na seguinte entrada:

192.168.0.192	20.0.0.18	255.255.255.240	UG	0 0	0 eth1
---------------	-----------	-----------------	----	-----	--------

Figura 2.1.22: Entrada da tabela de encaminhamento do *router n3* da rede *core* fornecida no enunciado

(Nota: Correção do *Gateway* da figura 2.1.22 para 10.0.0.18)

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.

Após a análise da entrada da figura 2.1.22, verificamos que a máscara usada nesta entrada é /28.

Com esta máscara, os endereços que possivelmente dão match pertencem ao intervalo de 192.168.0.192 até 192.168.0.207 podendo haver um match nesta entrada para os pacotes enviados para o polo *Galiza* e para parte do polo *CDN*. Se analisarmos o restante da tabela de endereçamento, verificamos que está entrada não é utilizada para o envio de pacotes para o polo *Galiza* e *CDN*.

Para o polo *Galiza* é utilizada a entrada cujo destino é 192.168.0.192 e máscara 255.255.255.248 pois está é a entrada com o maior match após aplicar a máscara. Já para o polo *CDN* as entradas utilizadas são as com destino: 192.168.0.200, 192.168.0.208 e 192.168.0.216. Estas com a mesma máscara, 255.255.255.248.

Se hipoteticamente esta entrada fosse utilizada não iríamos obter o funcionamento esperado do dispositivo, pois este iria enviar os pacotes para o *route n4* da rede *core* que iriam ser posteriormente enviados novamente para o *router n3* e assim sucessivamente nunca chegando ao destino.

2.1.6 Alínea f)

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 privados, pois os seus endereços de *Ip* encontram-se no intervalo de [192.168.0.0 - 192.168.255.255]. Relativamente à rede *core* também é uma rede privada, devido ao seu endereço de *Ip* localizar-se no intervalo [10.0.0.0 - 10.255.255.255].

2.1.7 Alínea g)

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

Os *switches* não têm um endereço de *IP* atribuído, por serem um dispositivo pertencente ao nível 2 da pilha protocolar, sendo os endereços *IP* pertencentes ao nível 3.

2.2 Pergunta 2

Tendo feito as pazes com a mãe, D. Afonso Henriques vê-se com algum tempo livre e decide fazer remodelações no condado:

2.2.1 Alínea a)

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. Explique ainda a utilidade de uma rota default.

Após executarmos o comando *"netstat -rn"* verificamos a seguinte tabela de encaminhamento na figura 2.2.1.

```
root@Castelo:/tmp/pycore.39239/Castelo.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@Castelo:/tmp/pycore.39239/Castelo.conf# █
```

Figura 2.2.1: Tabela de Encaminhamento do *host Castelo*.

Para eliminar a rota *default* do *Castelo*, executamos o comando *"route del default"*, e obtendo a seguinte tabela de encaminhamento, figura 2.2.2

```
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.39239/Castelo.conf# █
```

Figura 2.2.2: Tabela de Encaminhamento do *host Castelo* sem a sua rota *default*.

Começamos por garantir a ligação com o polo *Galiza*, utilizando o comando *"route add -net 192.168.0.192 netmask 255.255.255.248 gw 192.168.0.225"*, como podemos observar na tabela da figura 2.2.3.

```
root@Castelo:/tmp/pycore.39227/Castelo.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window irtt Iface
192.168.0.192  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
root@Castelo:/tmp/pycore.39227/Castelo.conf# []
```

Figura 2.2.3: Tabela de Encaminhamento do *host Castelo* com ligação à rede *Galiza*.

Posteriormente, estabelecemos ligação com o polo *Institucional*, utilizando os comandos:

- *"route add -net 192.168.0.232 netmask 255.255.255.248 gw 192.168.0.225"*

- ”route add -net 192.168.0.240 netmask 255.255.255.248 gw 192.168.0.225”
- ”route add -net 192.168.0.248 netmask 255.255.255.248 gw 192.168.0.225”

Como podemos observar na figura 2.2.4

```
root@Castelo:/tmp/pycore.39227/Castelo.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
192.168.0.192  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  255.255.255.248 UG       0 0          0 eth0
192.168.0.240  192.168.0.225  255.255.255.248 UG       0 0          0 eth0
192.168.0.248  192.168.0.225  255.255.255.248 UG       0 0          0 eth0
root@Castelo:/tmp/pycore.39227/Castelo.conf#
```

Figura 2.2.4: Tabela de Encaminhamento do host *Castelo* com ligação à rede *Institucional*.

Por fim, estabelecemos ligação com o polo *CDN*, utilizando os comandos:

- ”route add -net 192.168.0.200 netmask 255.255.255.248 gw 192.168.0.225”
- ”route add -net 192.168.0.208 netmask 255.255.255.248 gw 192.168.0.225”
- ”route add -net 192.168.0.216 netmask 255.255.255.248 gw 192.168.0.225”

Após adicionar todas as rotas para todas as redes da topologia, obtemos a tabela de encaminhamento da figura 2.2.5

```
root@Castelo:/tmp/pycore.39239/Castelo.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
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  255.255.255.248 UG       0 0          0 eth0
192.168.0.240  192.168.0.225  255.255.255.248 UG       0 0          0 eth0
192.168.0.248  192.168.0.225  255.255.255.248 UG       0 0          0 eth0
root@Castelo:/tmp/pycore.39239/Castelo.conf#
```

Figura 2.2.5: Tabela de Encaminhamento do host *Castelo* final.

Através do comando *ping*, verificamos a existência de conetividade entre o dispositivo *Castelo* e os restantes polos, como podemos verificar nas figuras 2.2.6 2.2.7

```
root@Castelo:/tmp/pycore.41017/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=0.067 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=61 time=0.165 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=61 time=0.072 ms
^C
--- 192.168.0.234 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2036ms
rtt min/avg/max/mdev = 0.067/0.101/0.165/0.045 ms
root@Castelo:/tmp/pycore.41017/Castelo.conf# ping 192.168.0.242
PING 192.168.0.242 (192.168.0.242) 56(84) bytes of data.
64 bytes from 192.168.0.242: icmp_seq=1 ttl=61 time=0.066 ms
64 bytes from 192.168.0.242: icmp_seq=2 ttl=61 time=0.085 ms
64 bytes from 192.168.0.242: icmp_seq=3 ttl=61 time=0.192 ms
^C
--- 192.168.0.242 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2037ms
rtt min/avg/max/mdev = 0.066/0.114/0.192/0.055 ms
root@Castelo:/tmp/pycore.41017/Castelo.conf# ping 192.168.0.251
PING 192.168.0.251 (192.168.0.251) 56(84) bytes of data.
64 bytes from 192.168.0.251: icmp_seq=1 ttl=61 time=0.069 ms
64 bytes from 192.168.0.251: icmp_seq=2 ttl=61 time=0.164 ms
64 bytes from 192.168.0.251: icmp_seq=3 ttl=61 time=0.150 ms
^C
--- 192.168.0.251 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2035ms
rtt min/avg/max/mdev = 0.069/0.127/0.164/0.041 ms
root@Castelo:/tmp/pycore.41017/Castelo.conf# █
```

Figura 2.2.6: *Ping* do dispositivo *Castelo* para os dispositivos do polo *Institucional*

```

root@Castelo:/tmp/pycore.41017/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=0.132 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=55 time=0.329 ms
^C
--- 192.168.0.194 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1005ms
rtt min/avg/max/mdev = 0.132/0.230/0.329/0.098 ms
root@Castelo:/tmp/pycore.41017/Castelo.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=0.318 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=55 time=0.361 ms
^C
--- 192.168.0.218 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1038ms
rtt min/avg/max/mdev = 0.318/0.339/0.361/0.021 ms
root@Castelo:/tmp/pycore.41017/Castelo.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=0.105 ms
64 bytes from 192.168.0.210: icmp_seq=2 ttl=55 time=0.116 ms
64 bytes from 192.168.0.210: icmp_seq=3 ttl=55 time=0.148 ms
^C
--- 192.168.0.210 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2055ms
rtt min/avg/max/mdev = 0.105/0.123/0.148/0.018 ms
root@Castelo:/tmp/pycore.41017/Castelo.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=0.229 ms
64 bytes from 192.168.0.203: icmp_seq=2 ttl=55 time=0.282 ms
64 bytes from 192.168.0.203: icmp_seq=3 ttl=55 time=0.272 ms
^C
--- 192.168.0.203 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2045ms
rtt min/avg/max/mdev = 0.229/0.261/0.282/0.023 ms
root@Castelo:/tmp/pycore.41017/Castelo.conf# []

```

Figura 2.2.7: *Ping* do dispositivo *Castelo* para os dispositivos do Polo *Galiza* e *CDN*

Uma rota “*default*” é utilizada quando não existe uma rota específica definida para um determinado destino. A sua utilidade é garantir que o tráfego é sempre encaminhado mesmo que não haja uma rota correspondente para o destino especificado.

2.2.2 Alínea b)

Por modo a garantir uma posição estratégicamente 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 es-

quema 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 número do nosso grupo prático é o *PL41*, logo, o endereço de rede atribuído pelo *ISP* é *172.16.41.128/26*, obtendo o seguinte valor em binário:

10101100.00010000.00101001.10000000

Destes 32 *bits*, os 26 primeiros são imutáveis, obtemos então endereços de *IP* no intervalo de *172.16.41.128* até *172.16.41.191*.

Sabendo que precisamos de um esquema de endereçamento de pelo menos 3 redes, onde cada uma garantam 10 ou mais *hosts*, precisamos de pelo menos 4 *bits* para representar os endereços dos *hosts* ($2^4 = 16$), destes 16 endereços sabemos que dois são reservados para o endereço de rede e para o endereço de *broadcast* restando 14 endereços para representar *hosts*. Obtemos assim, apenas dois *bits* para representar as sub-redes ($2^2 = 4$).

Calculamos então a nova máscara de rede, $26 + 2 = 28$, obtendo então o seguinte esquema de endereçamento da tabela 2.1.

Tabela 2.1: Esquema de endereçamento para o endereço *172.16.41.128/26*

Sub-rede	Endereço de rede	Hosts disponíveis	Endereço de Broadcast
Sub-rede 1	<i>172.16.41.128/28</i>	<i>172.16.41.129/28</i> até <i>172.16.41.142/28</i>	<i>172.16.41.143/28</i>
Sub-rede 2	<i>172.16.41.144/28</i>	<i>172.16.41.145/28</i> até <i>172.16.41.158/28</i>	<i>172.16.41.159/28</i>
Sub-rede 3	<i>172.16.41.160/28</i>	<i>172.16.41.161/28</i> até <i>172.16.41.174/28</i>	<i>172.16.41.175/28</i>
Sub-rede 4	<i>172.16.41.176/28</i>	<i>172.16.41.177/28</i> até <i>172.16.41.190/28</i>	<i>172.16.41.191/28</i>
TOTAL	4 sub-redes	14 <i>hosts</i> disponíveis por sub-rede	

Da tabela 2.1 obtemos então 4 sub-redes onde em cada uma é possível ter 14 *hosts*, concluindo assim que é um esquema de endereçamento válido.

2.2.3 Alínea c)

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?

Foi ligado um novo *host* diretamente ao router *ReiDaNet* atribuindo-lhe o endereço *172.16.41.140* da sub-rede 1 da tabela 2.1. Foi alterado o endereço da interface de ligação do router *ReiDaNet* para *172.16.41.129*, figura 2.2.8

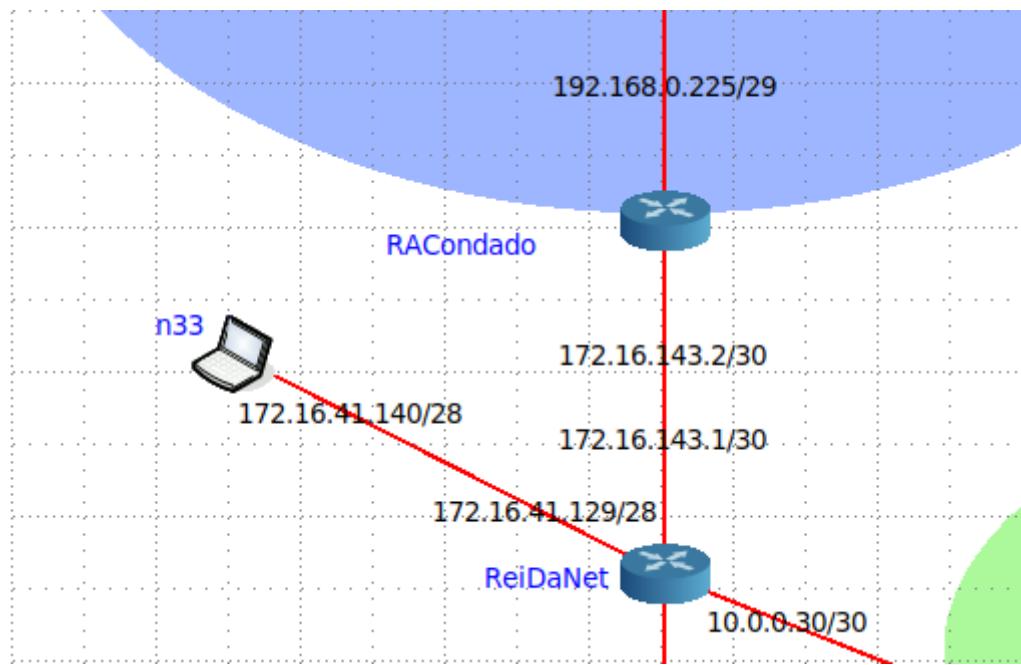


Figura 2.2.8: Ligação de um novo *host* ao *router ReiDaNet*.

Para verificar a existência de conectividade com os diferentes polos recorremos ao comando *"ping"*, figuras 2.2.9 e 2.2.10.

```

root@n33:/tmp/pycore.41017/n33.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=56 time=0.279 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=56 time=0.305 ms
64 bytes from 192.168.0.194: icmp_seq=3 ttl=56 time=0.263 ms
^C
--- 192.168.0.194 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2056ms
rtt min/avg/max/mdev = 0.263/0.282/0.305/0.017 ms
root@n33:/tmp/pycore.41017/n33.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=56 time=0.189 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=56 time=0.473 ms
64 bytes from 192.168.0.218: icmp_seq=3 ttl=56 time=0.273 ms
^C
--- 192.168.0.218 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.189/0.311/0.473/0.119 ms
root@n33:/tmp/pycore.41017/n33.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=56 time=0.381 ms
64 bytes from 192.168.0.210: icmp_seq=2 ttl=56 time=0.184 ms
64 bytes from 192.168.0.210: icmp_seq=3 ttl=56 time=0.117 ms
^C
--- 192.168.0.210 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2028ms
rtt min/avg/max/mdev = 0.117/0.227/0.381/0.112 ms
root@n33:/tmp/pycore.41017/n33.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=56 time=0.115 ms
64 bytes from 192.168.0.203: icmp_seq=2 ttl=56 time=0.226 ms
64 bytes from 192.168.0.203: icmp_seq=3 ttl=56 time=0.424 ms
^C
--- 192.168.0.203 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.115/0.255/0.424/0.127 ms
root@n33:/tmp/pycore.41017/n33.conf# []

```

Figura 2.2.9: Comando “ping” do host n33 para dispositivos dos Polos *CDN* e *Galiza*

```

root@n33:/tmp/pycore.41017/n33.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=0.124 ms
64 bytes from 192.168.0.226: icmp_seq=2 ttl=62 time=0.170 ms
64 bytes from 192.168.0.226: icmp_seq=3 ttl=62 time=0.200 ms
^C
--- 192.168.0.226 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 0.124/0.164/0.200/0.031 ms
root@n33:/tmp/pycore.41017/n33.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=62 time=0.072 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=62 time=0.117 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=62 time=0.170 ms
^C
--- 192.168.0.234 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2035ms
rtt min/avg/max/mdev = 0.072/0.119/0.170/0.040 ms
root@n33:/tmp/pycore.41017/n33.conf# ping 192.168.0.242
PING 192.168.0.242 (192.168.0.242) 56(84) bytes of data.
64 bytes from 192.168.0.242: icmp_seq=1 ttl=62 time=0.079 ms
64 bytes from 192.168.0.242: icmp_seq=2 ttl=62 time=0.205 ms
64 bytes from 192.168.0.242: icmp_seq=3 ttl=62 time=0.281 ms
^C
--- 192.168.0.242 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2045ms
rtt min/avg/max/mdev = 0.079/0.188/0.281/0.083 ms
root@n33:/tmp/pycore.41017/n33.conf# ping 192.168.0.251
PING 192.168.0.251 (192.168.0.251) 56(84) bytes of data.
64 bytes from 192.168.0.251: icmp_seq=1 ttl=62 time=0.060 ms
64 bytes from 192.168.0.251: icmp_seq=2 ttl=62 time=0.177 ms
64 bytes from 192.168.0.251: icmp_seq=3 ttl=62 time=0.156 ms
^C
--- 192.168.0.251 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2029ms
rtt min/avg/max/mdev = 0.060/0.131/0.177/0.050 ms
root@n33:/tmp/pycore.41017/n33.conf# []

```

Figura 2.2.10: Comando “ping” do host n33 para dispositivos dos Polos *Instutucional* e *Condado Portucalense*

Analisando as figuras 2.2.9 e 2.2.10, é possível verificar que existe conectividade entre o novo host e os diferentes Polos.

Porem, ao realizar o comando “ping” com o dispositivo *Castelo* do polo *Condado Portucalense* foram enviados 6 pacotes onde nenhum obteve resposta, como verificado pela figura 2.2.11

```

root@n33:/tmp/pycore.41017/n33.conf# ping 192.168.0.228
PING 192.168.0.228 (192.168.0.228) 56(84) bytes of data.
^C
--- 192.168.0.228 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5134ms

```

Figura 2.2.11: Comando “ping” do host *n33* para o dispositivo *Castelo* do Polo *Condado Portucalense*

Foi executado então o *wireshark* no dispositivo *castelo* com o objetivo de capturar tráfego *ICMP* enviado pelo comando “*traceroute -I*” executado no novo *host*, obtendo a captura da figura 2.2.12

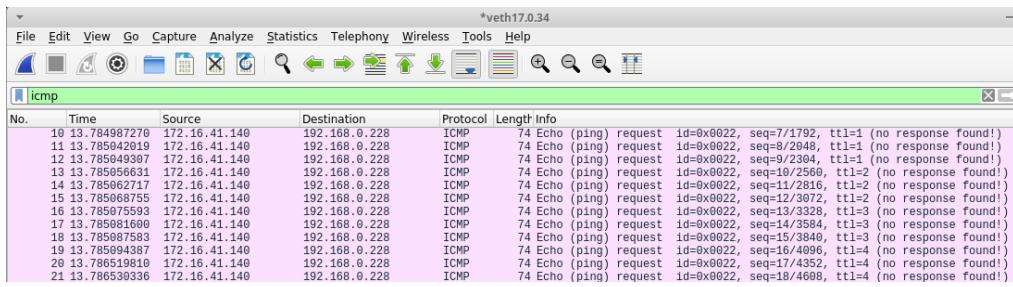


Figura 2.2.12: Captura de tráfego *ICMP* do dispositivo *Castelo* através do *wireshark*

Concluímos então que os pacotes enviados pelo novo *host* chegaram efetivamente ao dispositivo *Castelo*, porém, como foi removida a rota *default* da tabela de encaminhamento deste ultimo, na alínea a) subsecção 2.2.1, pela figura 2.2.5 observamos que não existe nenhuma entrada que indique por onde devem ser encaminhado os pacotes com destino à nova rede, não existindo conectividade entre os dois dispositivos.

2.3 Pergunta 3

Ao planear um novo ataque, D. Afonso Henriques 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 Alínea 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

Inicialmente, executamos o comando *"netstat -rn"* no dispositivo *n6* de modo a visualizar a sua tabela de encaminhamento, apresentada na figura 2.3.1.

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

Figura 2.3.1: Tabela de encaminhamento do dispositivo *n6*.

Eliminamos todas as rotas referentes a *Galiza* e *CDN* no dispositivo *n6*, como podemos observar na figura 2.3.2.

root@n6:/tmp/pycore.42811/n6.conf#	route del -net 192.168.0.192 netmask 255.255.255.248						
root@n6:/tmp/pycore.42811/n6.conf#	route del -net 192.168.0.200 netmask 255.255.255.248						
root@n6:/tmp/pycore.42811/n6.conf#	route del -net 192.168.0.208 netmask 255.255.255.248						
root@n6:/tmp/pycore.42811/n6.conf#	route del -net 192.168.0.216 netmask 255.255.255.248						
root@n6:/tmp/pycore.42811/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
192.168.0.224	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 2.3.2: Tabela de encaminhamento do dispositivo *n6* sem as rotas referentes a *Galiza* e *CDN*.

De seguida, para conseguirmos utilizar o *Supernetting*, foi necessário calcular a nova máscara a ser introduzida, para tal convertemos para binário os seguintes endereços de *IP*:

- $192.168.0.192 \Rightarrow \underline{11000000.10101000.00000000.11000000}$
- $192.168.0.200 \Rightarrow \underline{11000000.10101000.00000000.11001000}$
- $192.168.0.208 \Rightarrow \underline{11000000.10101000.00000000.11010000}$
- $192.168.0.216 \Rightarrow \underline{11000000.10101000.00000000.11011000}$

Após feita a conversão, verificamos quais os *bits* em comum nos vários endereços, estando eles identificados a **negrito**. De seguida, convertemos todos os *bits* em comum para "1" e os restantes para "0".

11111111.11111111.11111111.11100000

Obtendo a seguinte máscara de rede:

255.255.255.224

Por fim, é necessário calcular o novo endereço de rede. Para tal, mantemos os *bits* em comum e alteramos os incomuns para "0"

11000000.10101000.00000000.11000000

Obtendo o seguinte resultado:

192.168.0.192

Com todos os dados recolhidos, executamos o comando ”*add - net 192.168.0.192 netmask 255.255.255.224 gw 10.0.0.1*”.

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 2.3.3: Tabela de encaminhamento com a utilização do *Supernetting* para a *Galiza* e *CDN*

Por fim, falta verificar se a conetividade é mantida, para tal executamos o comando ”*ping 192.168.0.194*” no dispositivo *AfonsoHenriques*, de modo a saber se existia ligação com o dispositivo *Teresa* que se localiza no polo *Galiza*, no qual verificamos que a conexão era mantida, como podemos observar na seguinte figura 2.3.4.

```
Kycore.42811/AfonsoHenriques.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=0.216 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=55 time=0.242 ms
64 bytes from 192.168.0.194: icmp_seq=3 ttl=55 time=0.253 ms
64 bytes from 192.168.0.194: icmp_seq=4 ttl=55 time=0.260 ms
64 bytes from 192.168.0.194: icmp_seq=5 ttl=55 time=0.251 ms
...
```

Figura 2.3.4: Ligação estabelecida entre os dispositivos *AfonsoHenriques* e *Teresa*

Sendo também executado o comando ”*ping 192.169.0.218*” no dispositivo *AfonsoHenriques*, com o mesmo intuito de saber se existia ligação com o dispositivo *Spotify* que se localiza no polo *CDN*, no qual verificamos que a conexão também era mantida, como podemos observar na seguinte figura 2.3.5.

```
<pycore.42811/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=0.277 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=55 time=0.241 ms
64 bytes from 192.168.0.218: icmp_seq=3 ttl=55 time=0.242 ms
64 bytes from 192.168.0.218: icmp_seq=4 ttl=55 time=0.085 ms
64 bytes from 192.168.0.218: icmp_seq=5 ttl=55 time=0.290 ms
```

Figura 2.3.5: Ligação estabelecida entre os dispositivos *AfonsoHenriques* e *Spotify*

2.3.2 Alínea b)

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

Repetindo o processo, começamos por eliminar todas as rotas referentes ao *Condado Portucalense* e *Institucional* no dispositivo *n6*, como podemos observar na figura 2.3.6.

```
root@n6:/tmp/pycore.42811/n6.conf# route del -net 192.168.0.224 netmask 255.255.255.248
root@n6:/tmp/pycore.42811/n6.conf# route del -net 192.168.0.232 netmask 255.255.255.248
root@n6:/tmp/pycore.42811/n6.conf# route del -net 192.168.0.240 netmask 255.255.255.248
root@n6:/tmp/pycore.42811/n6.conf# route del -net 192.168.0.248 netmask 255.255.255.248
```

Figura 2.3.6: Eliminação das rotas referentes ao *Condado Portucalense* e *Institucional* no dispositivo *n6*

De seguida, calculamos a nova máscara a ser utilizada e para isso, convertemos para binário os seguintes endereços de *IP*:

- *192.168.0.224* => 11000000.10101000.00000000.11100000
- *192.168.0.232* => 11000000.10101000.00000000.11101000
- *192.168.0.240* => 11000000.10101000.00000000.11110000
- *192.168.0.248* => 11000000.10101000.00000000.11111000

Após a conversão, utilizamos novamente o mesmo método da alínea anterior.

11000000.10101000.00000000.11100000

Obtendo a seguinte máscara de rede:

255.255.255.224

Por fim, calculamos o novo endereço de rede utilizando novamente a abordagem da alínea anterior.

11000000.10101000.00000000.11100000

Obtendo o seguinte resultado:

192.168.0.224

Após a recolha dos dados necessários, executamos o comando ”*add - net 192.168.0.224 netmask 255.255.255.224 gw 10.0.0.6*” de modo a poder utilizar o *Supernetting*. Obtendo a seguinte tabela 2.3.7.

```
root@n6:/tmp/pycore.42811/n6.conf# route add -net 192.168.0.224 netmask 255.255.255.224 gw 10.0.0.6
root@n6:/tmp/pycore.42811/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      255.255.255.224 UG   0 0        0 eth0
192.168.0.224     10.0.0.6      255.255.255.224 UG   0 0        0 eth1
```

Figura 2.3.7: Tabela de encaminhamento com a utilização do *Supernetting* para o *Condado Portucalense e Institucional*

Por fim, verificamos a conectividade para os polos referentes, para isso executamos o comando ”*ping 192.168.0.228*” no dispositivo *Teresa*, de modo a saber se existia ligação com o dispositivo *Castelo*, que se localiza no *Condado Portucalense*, no qual verificamos que a conexão era mantida, como podemos observar na figura 2.3.8.

```
root@Teresa:/tmp/pycore.42811/Teresa.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=55 time=0.184 ms
64 bytes from 192.168.0.228: icmp_seq=2 ttl=55 time=0.311 ms
64 bytes from 192.168.0.228: icmp_seq=3 ttl=55 time=0.158 ms
64 bytes from 192.168.0.228: icmp_seq=4 ttl=55 time=0.256 ms
64 bytes from 192.168.0.228: icmp_seq=5 ttl=55 time=0.245 ms
```

Figura 2.3.8: Ligação estabelecida entre os dispositivos *Teresa* e *Castelo*.

Sendo também executado o comando ”*ping 192.169.0.234*” no dispositivo *Teresa*, com o mesmo intuito de saber se existia ligação com o dispositivo *UMinho* que se localiza no polo *Institucional*, no qual verificamos que a conexão também era mantida, como podemos observar na seguinte figura 2.3.9.

```
root@Teresa:/tmp/pycore.42811/Teresa.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=55 time=0.167 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=55 time=0.246 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=55 time=0.311 ms
64 bytes from 192.168.0.234: icmp_seq=4 ttl=55 time=0.241 ms
64 bytes from 192.168.0.234: icmp_seq=5 ttl=55 time=0.252 ms
```

Figura 2.3.9: Ligação estabelecida entre os dispositivos *Teresa* e *UMinho*.

2.3.3 Alínea c)

Comente os aspectos positivos e negativos do uso do Supernetting.

O *Supernetting* é o processo de combinar vários endereços de sub-redes, num só endereço. Este processo pode ser vantajoso devido a reduzir o tamanho de uma tabela de encaminhamento(Reduzindo também a memória ocupada) e por consequência diminuir o tempo de procura de um endereço na tabela e reduzindo a sobrecarga de um *router*. Em contraste, este processo pode não ser muito vantajoso, visto a aumentar a vulnerabilidade de uma rede em caso de ataque, que por consequência dá-se num aumento de problemas de segurança. Também é necessário haver um maior gestão e manutenção com o aumento de uma sub-rede.

Capítulo 3

Conclusões

Parte I

Na realização da primeira parte deste trabalho prático tivemos oportunidade de usar ferramentas como o *Core* e o *Wireshark*. Conseguimos consolidar, num ambiente prático, a matéria lecionada nas aulas teóricas sobre os protocolos *ICMP*, *IP* e sobre a fragmentação de pacotes.

Acreditamos vivamente ter conseguido superar todas as dificuldades com sucesso e alcançando os objetivos propostos.

Parte II

Em suma, ao realizar esta segunda parte do trabalho prático, assimilamos as diferenças entre endereços públicos e privados, o funcionamento das tabelas de endereçamento, a utilização do comando "*ping*" e "*traceroute -I*" para testar a conectividade entre dispositivos em cenários diferentes, a adição e remoção de rotas através dos comandos "*route add -net endereço-ip netmask máscara gw próximo-salto*" e "*route del -net endereço-ip netmask máscara*", e por fim, sobre o funcionamento de *subnetting* e utilização do *supernetting*.

Acreditamos ter superado todas as dificuldades encontradas neste trabalho prático, alcançando assim todos os objetivos propostos.