
Trabalho Prático Nº2 – Protocolo IPv4

TRABALHO REALIZADO POR:

BEATRIZ RIBEIRO MONTEIRO
CARLOS EDUARDO CULOLO DANTAS DA COSTA
HUGO RICARDO MACEDO GOMES



A95437
Beatriz Monteiro



A88551
Carlos Costa



A96842
Hugo Gomes

PL5 GRUPO 55
REDES DE COMPUTADORES 22/23
UNIVERSIDADE DO MINHO

Índice

1	Introdução	1
2	Parte 1 – Protocolo IPv4 :: Datagramas IP e Fragmentação	2
2.1	Exercício 1	2
2.1.1	Alínea a	2
2.1.2	Alínea b	3
2.1.3	Alínea c	3
2.1.4	Alínea d	4
2.2	Exercício 2	5
2.2.1	Alínea a	5
2.2.2	Alínea b	6
2.2.3	Alínea c	6
2.2.4	Alínea d	6
2.2.5	Alínea e	7
2.2.6	Alínea f	7
2.2.7	Alínea g	8
2.2.8	Alínea h	9
2.3	Exercício 3	10
2.3.1	Alínea a	10
2.3.2	Alínea b	10
2.3.3	Alínea c	10
2.3.4	Alínea d	11
2.3.5	Alínea e	11
2.3.6	Alínea f	12
2.3.7	Alínea g	12
2.3.8	Alínea h	12
2.3.9	Alínea i	12
2.3.10	Alínea j	13
3	Parte 2 – Protocolo IPv4:: Endereçamento e Encaminhamento IP	14
3.1	Exercício 1	14
3.1.1	Alínea a	14
3.1.2	Alínea b	16
3.1.3	Alínea c	16
3.1.4	Alínea d	18
3.1.5	Alínea e	19
3.1.6	Alínea f	20
3.1.7	Alínea g	20
3.2	Exercício 2	21
3.2.1	Alínea a	21
3.2.2	Alínea b	23
3.2.3	Alínea c	24
3.3	Exercício 3	25
3.3.1	Alínea a	25
3.3.2	Alínea b	25
3.3.3	Alínea c	26
4	Conclusão	27

List of Figures

1	Topologia	2
2	Tráfego ICMP enviado pelo sistema <i>Lost</i> e o tráfego ICMP recebido como resposta	3
3	Comando <i>traceroute -I</i> no <i>host</i> <i>Lost</i> para o endereço IP do <i>Found</i>	4
4	Primeira mensagem ICMP capturada	5
5	Primeira mensagem ICMP capturada	6
6	Pacotes ordenados	7
7	Pacote 5	7
8	Pacote 81	7
9	Pacote 7	8
10	Série de Respostas ICMP TTL Exceeded	8
11	Pacote	9
12	Pacote 42	9
13	Pacote 8	9
14	Primeira mensagem ICMP	10
15	Segunda mensagem ICMP	11
16	Última mensagem ICMP	11
17	Não ocorre fragmentação.	13
18	Ocorre fragmentação.	13
19	Topologia da Rede	14
20	Conectividade com o servidor <i>Financas</i>	14
21	Conectividade com o servidor <i>HBO</i>	15
22	Conectividade com o servidor <i>Itunes</i>	15
23	Conectividade com o servidor <i>Netflix</i>	15
24	Conectividade com o servidor <i>Spotify</i>	15
25	Conectividade com o servidor <i>Youtube</i>	15
26	Tabela de encaminhamento no <i>AfonsoHenriques</i>	16
27	Tabela de encaminhamento na <i>Teresa</i>	16
28	Entrada na tabela de encaminhamento do <i>router</i> <i>n2</i>	17
29	Entrada na tabela de encaminhamento do <i>router</i> <i>n1</i>	17
30	Entrada na tabela de encaminhamento do <i>router</i> <i>n3</i>	17
31	Comando <i>traceroute</i> no <i>AfonsoHenriques</i> para a <i>Teresa</i>	18
32	Comando <i>ping</i> do <i>AfonsoHenriques</i> para a <i>Teresa</i>	18
33	Comando <i>ping</i> da <i>Teresa</i> para o <i>AfonsoHenriques</i>	18
34	Rota seguida pelo <i>Afonso</i>	19
35	Rota seguida pela <i>Teresa</i>	19
36	Comando de remoção da rota <i>default</i>	21
37	Tabela de encaminhamento do <i>Castelo</i>	21
38	Conectividade com o <i>host</i> <i>Teresa</i> do polo <i>Condado Portucalense</i>	22
39	Conectividade com o <i>host</i> <i>DI</i> do polo <i>Intitucional</i>	22
40	Conectividade com o <i>host</i> <i>UMinho</i> do polo <i>Intitucional</i>	22
41	Conectividade com o <i>host</i> <i>SegurancaSocial</i> do polo <i>Intitucional</i>	22
42	Conectividade com o <i>host</i> <i>Youtube</i> do polo <i>CDN</i>	22
43	Conectividade com o <i>host</i> <i>Spotify</i> do polo <i>CDN</i>	23
44	Conectividade com o <i>host</i> <i>Itunes</i> do polo <i>CDN</i>	23
45	Topologia alterada	24
46	Tabela de roteamento do <i>n6</i> com <i>Galiza</i> e <i>CDN</i>	25
47	Tabela de roteamento do <i>n6</i> com <i>CondadoPortucalense</i> e <i>Institucional</i>	25
48	Esquema do supernetting feito	26

1 Introdução

O presente relatório engloba todo o trabalho realizado nas duas partes do segundo trabalho prático proposto pela equipa docente da Unidade Curricular de Redes de Computadores.

O principal objetivo deste projeto é o estudo do *Internet Protocol* (IP), nomeadamente, o estudo do formato dos datagramas IP, da fragmentação de pacotes IP, dos endereços IP e do encaminhamento IP.

Na primeira parte, iremo-nos focar no registo de datagramas IP enviados e recebidos através da execução do programa *traceroute*.

Na segunda parte, continua-se o estudo do protocolo IPv4 com ênfase no endereçamento e encaminhamento IP. Serão estudadas algumas das técnicas mais relevantes que foram propostas para aumentar a escalabilidade do protocolo IP, mitigar a exaustão dos endereços IPv4 e também reduzir os recursos de memória necessários nos *routers* para manter as tabelas de encaminhamento

2 Parte 1 – Protocolo IPv4 :: Datagramas IP e Fragmentação

2.1 Exercício 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.

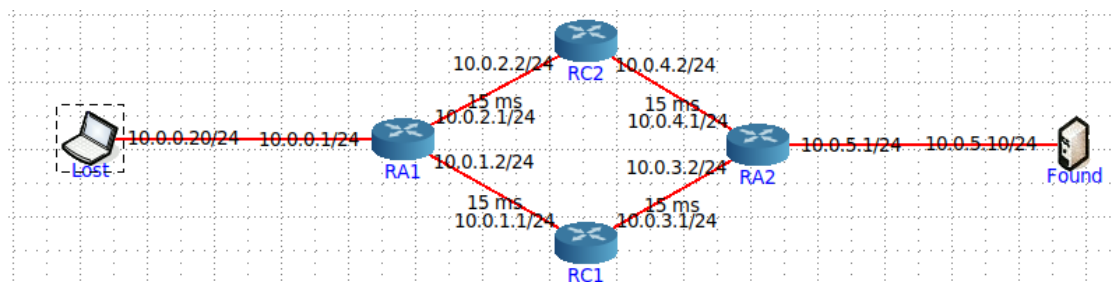


Figure 1: Topologia

2.1.1 Alínea a

Ative 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*.

O *traceroute* envia um ou mais datagramas com o campo TTL de vários valores desde o valor 1, 2, 3 e assim sucessivamente, sendo que estes pacotes são enviados para o mesmo destino. Uma vez que o percurso, desde o *Lost* (cliente) até ao *Found* (servidor), será decrementado 1 o TTL de cada datagrama recebido.

Sendo assim, podemos verificar pela Figura 1 que os datagramas enviados com valor TTL inferior ou igual a 3, não obtêm qualquer resposta, pois não chegam a *Found*. Só obtemos resposta, a partir de datagramas com valores TTL superiores e iguais a 4 e param o seu envio, até que o *Lost* receba uma resposta do *Found*.

22	22.935021832	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response found!)
23	22.935050236	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
24	22.935053792	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=2/512, ttl=1 (no response found!)
25	22.935059242	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
26	22.935073561	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response found!)
27	22.935078999	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
28	22.935082668	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no response found!)
29	22.935109659	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no response found!)
30	22.935114799	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no response found!)
31	22.935119158	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no response found!)
32	22.935122965	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no response found!)
33	22.935126712	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no response found!)
34	22.935130800	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in 53)
35	22.935134447	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in 54)
36	22.935137984	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (reply in 55)
37	22.935143153	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (reply in 56)
38	22.935147351	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (reply in 57)
39	22.935150988	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (reply in 58)
40	22.935156289	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (reply in 59)
41	22.939913331	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=17/4352, ttl=6 (reply in 60)
42	22.939024164	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=18/4608, ttl=6 (reply in 61)
43	22.939030987	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=19/4864, ttl=7 (reply in 62)
44	22.998439022	10.0.1.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
45	22.998446557	10.0.1.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
46	22.998447569	10.0.1.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
47	22.999487695	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=20/5120, ttl=7 (reply in 63)
48	22.999510027	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=21/5376, ttl=7 (reply in 64)
49	22.999516720	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=22/5632, ttl=8 (reply in 65)
50	23.029386705	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
51	23.029397175	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
52	23.029398417	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
53	23.029399710	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=10/2560, ttl=61 (request in 34)
54	23.029400762	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=11/2816, ttl=61 (request in 35)
55	23.029401954	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=12/3072, ttl=61 (request in 36)
56	23.029403156	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=13/3328, ttl=61 (request in 37)
57	23.029404379	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=14/3584, ttl=61 (request in 38)
58	23.029405431	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=15/3840, ttl=61 (request in 39)
59	23.029406473	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=16/4096, ttl=61 (request in 40)
60	23.029407675	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=17/4352, ttl=61 (request in 41)
61	23.029408727	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=18/4608, ttl=61 (request in 42)
62	23.029409899	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=19/4864, ttl=61 (request in 43)
63	23.061761163	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=20/5120, ttl=61 (request in 47)
64	23.061771753	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=21/5376, ttl=61 (request in 48)
65	23.061772735	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=22/5632, ttl=61 (request in 49)

Figure 2: Tráfego ICMP enviado pelo sistema *Lost* e o tráfego ICMP recebido como resposta

2.1.2 Alínea b

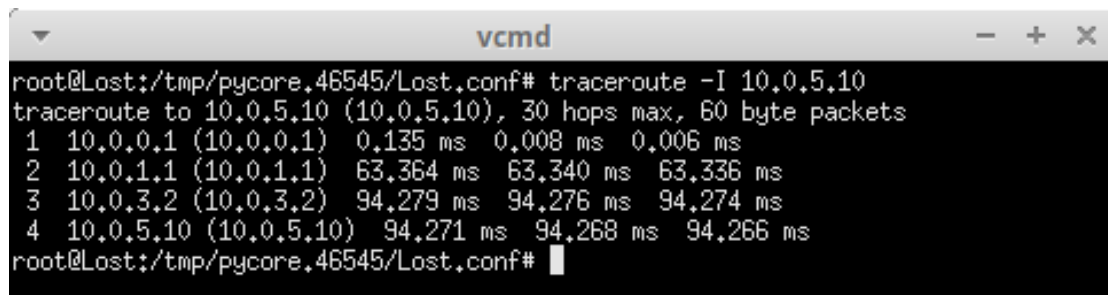
Qual deve ser o valor inicial mínimo do campo TTL para alcançar o servidor *Found*? Verifique na prática que a sua resposta está correta.

O valor inicial mínimo do campo TTL deve ser 4, uma vez que realiza a travessia por 3 *routers*. Se o valor de campo TTL fosse 3, então chegaria ao *router* de acesso RA2 com o valor 0, o que faria com que o datagrama fosse descartado, não chegando assim ao servidor *Found*. Podemos verificar isso na Figura 1, de forma em que os datagramas com valores TTL 4, recebe uma resposta, ao invés dos datagramas com valores TTL 3 que devolvem "no response found!".

2.1.3 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.

O valor médio do tempo de ida e volta (RTT) obtido no acesso ao servidor *Found* é 94,268 ms.



```
vcmd
root@Lost:/tmp/pycore.46545/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)  0.135 ms  0.008 ms  0.006 ms
 2 10.0.1.1 (10.0.1.1)  63.364 ms  63.340 ms  63.336 ms
 3 10.0.3.2 (10.0.3.2)  94.279 ms  94.276 ms  94.274 ms
 4 10.0.5.10 (10.0.5.10)  94.271 ms  94.268 ms  94.266 ms
root@Lost:/tmp/pycore.46545/Lost.conf#
```

Figure 3: Comando *traceroute -I* no *host* Lost para o endereço IP do Found

2.1.4 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?

O tempo que demora a ir da origem ao destino não é obrigatoriamente igual ao tempo que demora a voltar do destino à origem, na grande maioria dos casos é diferente. Para se fosse possível calcular valor do atraso num sentido seria preciso saber o tempo de chegada do pacote a cada um dos *routers* por onde passou, tornando esta tarefa bastante difícil.

2.2 Exercício 2

Pretende-se agora usar o *traceroute* na sua máquina nativa e gerar datagramas IP de diferentes tamanhos.

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.

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*)

5	1.016499	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=189/48384, ttl=1 (no resp...
6	1.018109	172.26.254.254	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
7	1.018549	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=190/48640, ttl=1 (no resp...
8	1.019389	172.26.254.254	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
9	1.020222	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=191/48896, ttl=1 (no resp...
10	1.021063	172.26.254.254	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
41	6.559318	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=192/49152, ttl=2 (no resp...
42	6.561452	172.26.254.254	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
43	6.561813	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=193/49408, ttl=2 (no resp...
44	6.564972	172.26.254.254	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
45	6.565338	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=194/49664, ttl=2 (no resp...
46	6.566707	172.26.254.254	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
79	12.097492	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=195/49920, ttl=3 (no resp...
80	12.099976	172.16.115.252	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
81	12.100648	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=196/50176, ttl=3 (no resp...
82	12.102327	172.16.115.252	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
83	12.102915	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=197/50432, ttl=3 (no resp...
84	12.104399	172.16.115.252	172.26.73.161	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
182	17.643970	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=198/50688, ttl=4 (reply i...
183	17.660198	193.136.9.240	172.26.73.161	ICMP	106 Echo (ping) reply id=0x0001, seq=198/50688, ttl=61 (reques...
184	17.661056	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=199/50944, ttl=4 (reply i...
185	17.672896	193.136.9.240	172.26.73.161	ICMP	106 Echo (ping) reply id=0x0001, seq=199/50944, ttl=61 (reques...
186	17.673845	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request id=0x0001, seq=200/51200, ttl=4 (reply i...
187	17.683796	193.136.9.240	172.26.73.161	ICMP	106 Echo (ping) reply id=0x0001, seq=200/51200, ttl=61 (reques...

Figure 4: Primeira mensagem ICMP capturada

2.2.1 Alínea a

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

Como é visível na figura 5, o endereço IP na interface ativa do computador que utilizamos é 172.26.73.161.

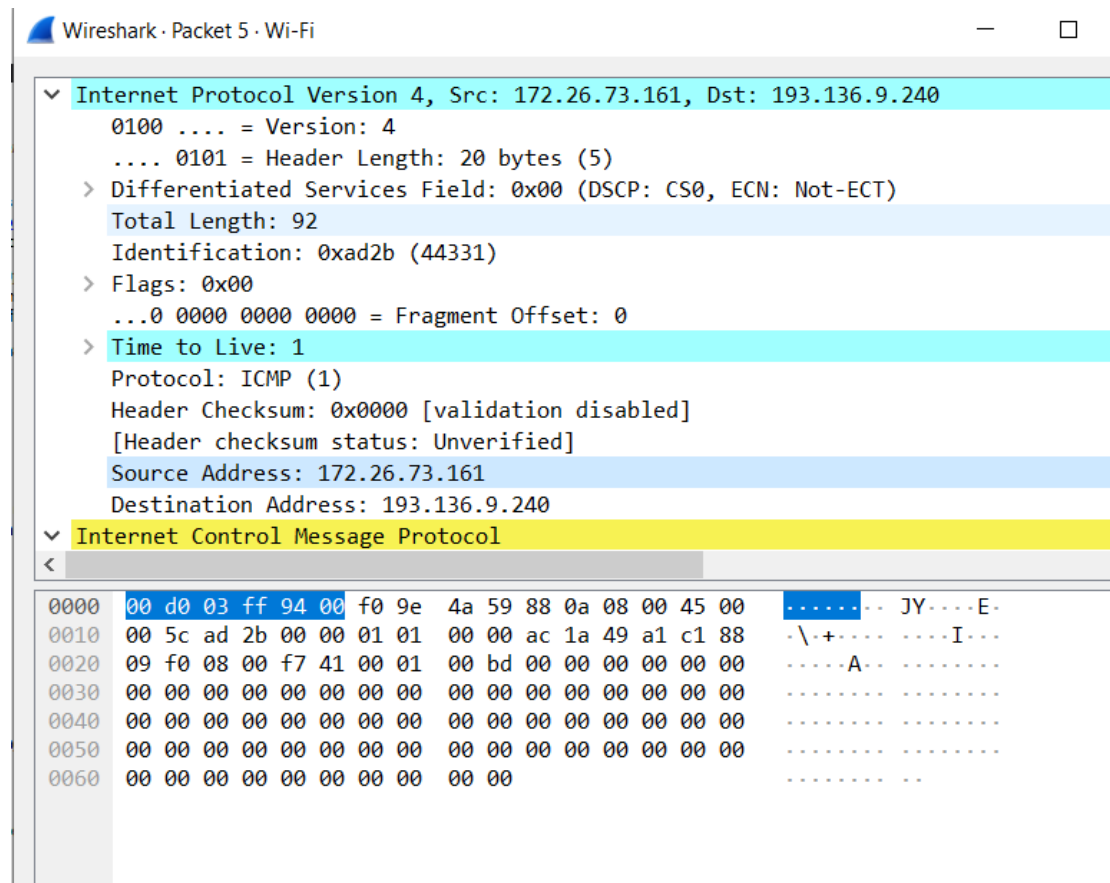


Figure 5: Primeira mensagem ICMP capturada

2.2.2 Alínea b

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

O valor do campo *protocol* é 01 que representa o protocolo ICMP (*Internet Control Message Protocol*).

2.2.3 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?

O cabeçalho IPv4 do datagrama (*Header Length*) tem 20 *bytes*, como é possível visualizar na figura 5. Podemos ainda verificar que o *Total Length* é de 92 *bytes*.

O *payload* tem, portanto, 72 *bytes*, uma vez que este é a diferença entre o *Total Length* (92 *bytes*) e o *Header Length* (20 *bytes*).

2.2.4 Alínea d

O datagrama IP foi fragmentado? Justifique.

Ainda na figura 5 é possível verificar que tanto o campo *fragment offset* como as flags são iguais a 0, logo, podemos concluir que o datagrama não foi fragmentado.

2.2.5 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.

5	1.016499	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=189/48384, ttl=1	(no response found!)
7	1.018549	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=190/48640, ttl=1	(no response found!)
9	1.020222	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=191/48896, ttl=1	(no response found!)
41	6.559318	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=192/49152, ttl=2	(no response found!)
43	6.561813	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=193/49408, ttl=2	(no response found!)
45	6.565338	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=194/49664, ttl=2	(no response found!)
79	12.097492	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=195/49920, ttl=3	(no response found!)
81	12.100648	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=196/50176, ttl=3	(no response found!)
83	12.102915	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=197/50432, ttl=3	(no response found!)
182	17.643970	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=198/50688, ttl=4	(reply in 183)
184	17.661056	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=199/50944, ttl=4	(reply in 185)
186	17.673845	172.26.73.161	193.136.9.240	ICMP	106 Echo (ping) request	id=0x0001, seq=200/51200, ttl=4	(reply in 187)
183	17.690108	193.136.9.240	172.26.73.161	ICMP	106 Echo (ping) reply	id=0x0001, seq=198/50688, ttl=61	(request in 182)
185	17.672896	193.136.9.240	172.26.73.161	ICMP	106 Echo (ping) reply	id=0x0001, seq=199/50944, ttl=61	(request in 184)
187	17.683796	193.136.9.240	172.26.73.161	ICMP	106 Echo (ping) reply	id=0x0001, seq=200/51200, ttl=61	(request in 186)

Figure 6: Pacotes ordenados

Após ordenarmos os pacotes de acordo com o endereço IP fonte e os analisarmos a percebermos que os únicos campos do cabeçalho IP que variavam era o TTL e a identificação. Deixamos dois datagramas que permiter ver essas diferenças.

```
Frame 5: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface \Device\NPF_{9C2697CA-2BE2-486A-9DF8-7B3848F99B25}, id 0
Ethernet II, Src: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.73.161, 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: 92
  Identification: 0xad2b (44331)
  Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. .. = More fragments: Not set
  Fragment offset: 0
  Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.26.73.161
  Destination: 193.136.9.240
```

Figure 7: Pacote 5

```
Frame 81: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface \Device\NPF_{9C2697CA-2BE2-486A-9DF8-7B3848F99B25}, id 0
Ethernet II, Src: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  Destination: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  Source: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 172.26.73.161, 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: 92
  Identification: 0xad32 (44338)
  Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. .. = More fragments: Not set
  Fragment offset: 0
  Time to live: 3
  Protocol: ICMP (1)
  Header checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.26.73.161
  Destination: 193.136.9.240
```

Figure 8: Pacote 81

2.2.6 Alínea f

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

É possível verificar alguns padrões após a análise do datagrama no campo da identificação e do TTL, estes padrões são visíveis nas figuras 6, 7 e 9.

O campo do TTL é incrementado a cada 3 tramas, isto porque foram enviadas 3 tramas para cada valor do TTL, enquanto o campo da identificação é incrementado com cada trama.

```

Frame 7: 106 bytes on wire (848 bits), 106 bytes captured (848 bits) on interface \Device\NPF_{9C2697CA-2BE2-486A-9DF8-7B3848F99B25}, id 0
Ethernet II, Src: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  Destination: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  Source: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a)
  Type: IPv4 (0x0800)
Internet Protocol Version 4, Src: 172.26.73.161, 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: 92
  Identification: 0xad2c (44332)
  Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0... .. = Don't fragment: Not set
    ..0... .. = More fragments: Not set
  Fragment offset: 0
  Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.26.73.161
  Destination: 193.136.9.240

```

Figure 9: Pacote 7

2.2.7 Alínea 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	Proto	Length	Info
6	1.018109	172.26.254.254	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
8	1.019880	172.26.254.254	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10	1.021963	172.26.254.254	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
42	6.561452	172.16.2.1	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
44	6.564972	172.16.2.1	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
46	6.566707	172.16.2.1	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
80	12.099976	172.16.115.252	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
82	12.102327	172.16.115.252	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
84	12.104399	172.16.115.252	172.26.73.161	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
183	17.660108	193.136.9.240	172.26.73.161	ICMP	106	Echo (ping) reply id=0x0001, seq=198/50688, ttl=61 (re...
185	17.672896	193.136.9.240	172.26.73.161	ICMP	106	Echo (ping) reply id=0x0001, seq=199/50944, ttl=61 (re...
187	17.683796	193.136.9.240	172.26.73.161	ICMP	106	Echo (ping) reply id=0x0001, seq=200/51200, ttl=61 (re...

Figure 10: Série de Respostas ICMP TTL Exceeded

- 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?

Os valores do campo TTL recebidos são 255,254,253, como é possível visualizar nas figuras 11, 12 e 13. Este é decrementado à medida que faz mais hops. O valor de TTL que recebemos é o valor do TTL quando chega ao respetivo ponto.

```

Frame 80: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C2697CA-2BE2-486A-9DF8-7B3848F99B25}, id 0
Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a)
Internet Protocol Version 4, Src: 172.16.115.252, Dst: 172.26.73.161
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x9db6 (40382)
  > Flags: 0x0000
    Fragment offset: 0
  Time to live: 253
  Protocol: ICMP (1)
  Header checksum: 0x0a3e [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.16.115.252
  Destination: 172.26.73.161

```

Figure 11: Pacote 80

```

> Frame 42: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C2697CA-2BE2-486A-9DF8-7B3848F99B25}, id 0
> Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a)
> Internet Protocol Version 4, Src: 172.16.2.1, Dst: 172.26.73.161
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x06b1 (1713)
  > Flags: 0x0000
    Fragment offset: 0
  Time to live: 254
  Protocol: ICMP (1)
  Header checksum: 0x1247 [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.16.2.1
  Destination: 172.26.73.161

```

Figure 12: Pacote 42

```

> Frame 8: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C2697CA-2BE2-486A-9DF8-7B3848F99B25}, id 0
> Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: f0:9e:4a:59:88:0a (f0:9e:4a:59:88:0a)
> Internet Protocol Version 4, Src: 172.26.254.254, Dst: 172.26.73.161
  0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 56
    Identification: 0xc7c9 (51145)
  > Flags: 0x0000
    Fragment offset: 0
  Time to live: 255
  Protocol: ICMP (1)
  Header checksum: 0x5266 [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.26.254.254
  Destination: 172.26.73.161

```

Figure 13: Pacote 8

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

Para fazer o maior número de *hops* possíveis para chegar ao ponto.

2.2.8 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?

Se as informações do cabeçalho ICMP fossem incluídas no cabeçalho IPv4 poderia existir algumas vantagens em termos de eficiência de rede. Por exemplo, esta inclusão reduziria o tamanho dos pacotes de rede, o que poderia melhorar o desempenho em redes de baixa largura de banda ou com alta taxa de perda de pacotes.

Porém existem algumas desvantagens significativas em incluir informações do cabeçalho ICMP no cabeçalho IPv4, como, por exemplo, isto tornaria o cabeçalho IPv4 mais complexo e difícil de ser analisado. Para além disso, se uma mensagem ICMP tiver que ser enviada, isso exigiria que todo o cabeçalho IPv4 fosse reenviado, o que poderia levar a uma carga adicional na rede.

2.3 Exercício 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). De modo a poder visualizar os fragmentos, aceda a *Edit -> Preferences -> Protocols* e em IPv4 desative a opção “*Reassemble fragmented IPv4 datagrams*”.

2.3.1 Alínea a

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

Como é visível na figura 14, o *Total Length* do datagrama é de 1500 *bytes*, enquanto que o pacote que pretendemos enviar tem 3555 *bytes* e, por isso, é necessário que este seja fragmentado em 3 *chunks* para ser enviado.

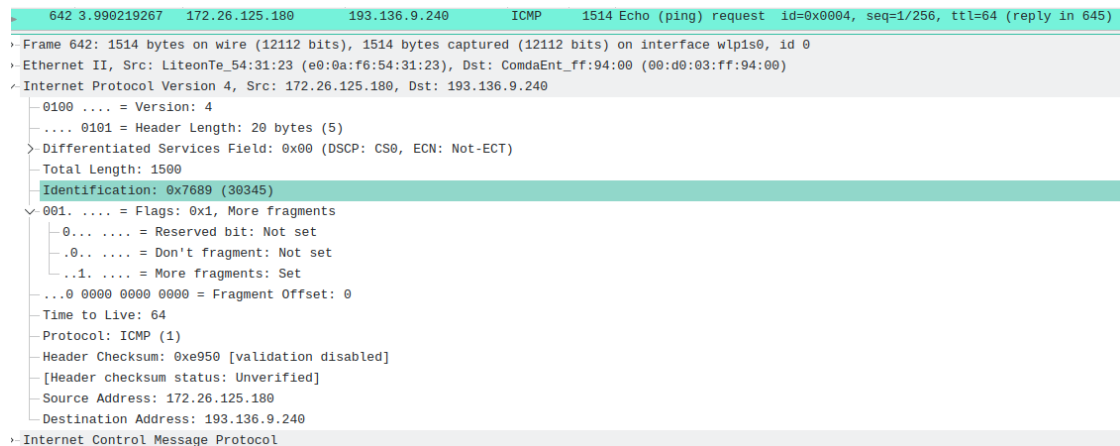


Figure 14: Primeira mensagem ICMP

2.3.2 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?

Os campos que permitem identificar que o datagrama foi fragmentado são: a *flag More Fragments*, o *Fragment Offset* e a identificação do pacote.

Na figura 14 conseguimos ver o datagrama IP do primeiro fragmento, conseguimos perceber que, de facto se trata do primeiro fragmento pois a *flag More Fragments* tem o valor 1, o que indica que existem mais fragmentos e tem o campo *Fragments Offset* a 0. É ainda visível que o tamanho do datagrama é de 1500 *bytes* (enviando 1480 *bytes* de informação).

2.3.3 Alínea c

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

O *Fragment Offset* é de 1480, o que nos permite concluir que este não é o primeiro fragmento, também podemos ver que não é o último porque o a *flag More Fragments* é 1, existindo, desta forma, mais fragmentos.

```

643 3.990241546 172.26.125.180 193.136.9.240 IPv4 1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=7689)
Frame 643: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp1s0, id 0
Ethernet II, Src: LiteonTe_54:31:23 (e0:0a:f6:54:31:23), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.125.180, 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: 0x7689 (30345)
  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
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0xe897 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.125.180
  Destination Address: 193.136.9.240

```

Figure 15: Segunda mensagem ICMP

2.3.4 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*.

Teoricamente deveriam ser gerados 3 fragmentos, 2 com 1480 *bytes* de informação e o último com 595 *bytes*. No entanto, como é visível na figura 16, este último fragmento tem 623 *bytes*, sendo 603 *bytes* de informação.

```

644 3.990245946 172.26.125.180 193.136.9.240 IPv4 637 Fragmented IP protocol (proto=ICMP 1, off=2960, ID=7689)
Frame 644: 637 bytes on wire (5096 bits), 637 bytes captured (5096 bits) on interface wlp1s0, id 0
Ethernet II, Src: LiteonTe_54:31:23 (e0:0a:f6:54:31:23), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.125.180, 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: 623
  Identification: 0x7689 (30345)
  000. .... = Flags: 0x0
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  ...0 0001 0111 0010 = Fragment Offset: 2960
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0x0b4c [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.125.180
  Destination Address: 193.136.9.240

```

Figure 16: Última mensagem ICMP

2.3.5 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.

O último fragmento tem a *flag More Fragments* a 0 e o *Fragment Offset* é diferente

de 0. O filtro que teríamos que aplicar seria : `ip.flags.mf == 0 && ip.frag_offset != 0`.

2.3.6 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, a partir dos seus fragmentos no *host* destino, que, a partir dos campos que já foram mencionados anteriormente junta os fragmentos pela ordem correta. Esta reconstrução pode ser feita em qualquer equipamento, desde que este seja o *host* destino deste datagrama.

2.3.7 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.

Entre os diferentes fragmentos os valores que mudam no cabeçalho IP são os valores correspondentes ao *Fragment Offset* e *More Fragments*. A *flag Fragment Offset* permite-nos saber qual a ordem que devemos organizar os fragmentos, uma vez que estes aparecem por ordem crescente e a *flag More Fragments* permite-nos averiguar se existem mais fragmentos do datagrama original. Juntando estas duas *flags* podemos reconstruir o datagrama original.

2.3.8 Alínea h

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

Os restantes fragmentos só enviam informação, sendo o primeiro fragmento o único com um protocolo ICMP, por isso este é o único a ser identificado como um pacote ICMP enquanto os outros são identificados como IPv4, pois só enviam dados.

2.3.9 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 do tamanho do datagrama com o qual será comparado, a fim de se determinar se o datagrama será fragmentado ou não, é o *Total Length* (ou MTU ?). O MTU , neste caso, tem um tamanho de 1500 *bytes*, em que 20 *bytes* fazem parte do cabeçalho.

No caso de aumentarmos o valor do *Total Length*, pode haver uma subutilização da rede, pois existe um espaço adicional no datagrama que poderia ser usado para transportar mais dados. A consequência disso seria levar a um desperdício de recursos da rede, reduzindo a sua eficiência. Se diminuirmos o valor do *Total Length*, levará a uma maior quantidade de fragmentos dos datagramas antes de ser transmitido pela rede. A consequência disso seria aumentar a sobrecarga da rede, pois seriam gastos mais recursos para fragmentar e reagrupar os fragmentos dos datagramas. Junto a isto, a elevada

fragmentação dos datagramas pode levar a problemas de desempenho, como aumento do atraso e da perda de fragmentos.

2.3.10 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?

Utilizando o comando "ping -M dont -s 1472 marco.uminho.pt" verificamos que não existe fragmentação, uma vez que apenas enviamos 1472 *bytes* onde 8 são de informação e 20 de cabeçalho ($1472 + 8 + 20 = 1500$), não havendo assim necessidade de fragmentação, como podemos verificar na figura 17.

The image shows a Wireshark packet capture of a ping command. The first packet is an ICMP Echo (ping) request from 172.26.39.80 to 193.136.9.240. The second packet is the corresponding ICMP Echo (ping) reply. The packet details for the request show an Ethernet II header, an Internet Protocol Version 4 header with a total length of 1500, and an ICMP Echo (ping) request. The flags field shows 'Don't fragment: Not set' and 'More fragments: Not set'. The fragment offset is 0. The packet is 1514 bytes on wire.

Figure 17: Não ocorre fragmentação.

A título de demonstração utilizamos o comando "ping -M dont -s 1473 marco.uminho.pt" e verificamos que existiu fragmentação, como podemos ver na figura 18.

The image shows a Wireshark packet capture of a ping command that results in fragmentation. The first packet is an ICMP Echo (ping) request from 172.26.39.80 to 193.136.9.240. The packet details show an Ethernet II header, an Internet Protocol Version 4 header with a total length of 1500, and an ICMP Echo (ping) request. The flags field shows 'Don't fragment: Not set' and 'More fragments: Set'. The fragment offset is 0. The packet is 1514 bytes on wire. The second packet is the corresponding ICMP Echo (ping) reply. The packet details for the request show an Ethernet II header, an Internet Protocol Version 4 header with a total length of 1500, and an ICMP Echo (ping) request. The flags field shows 'Don't fragment: Not set' and 'More fragments: Set'. The fragment offset is 0. The packet is 1514 bytes on wire.

Figure 18: Ocorre fragmentação.

3 Parte 2 – Protocolo IPv4:: Endereçamento e Encaminhamento IP

3.1 Exercício 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.

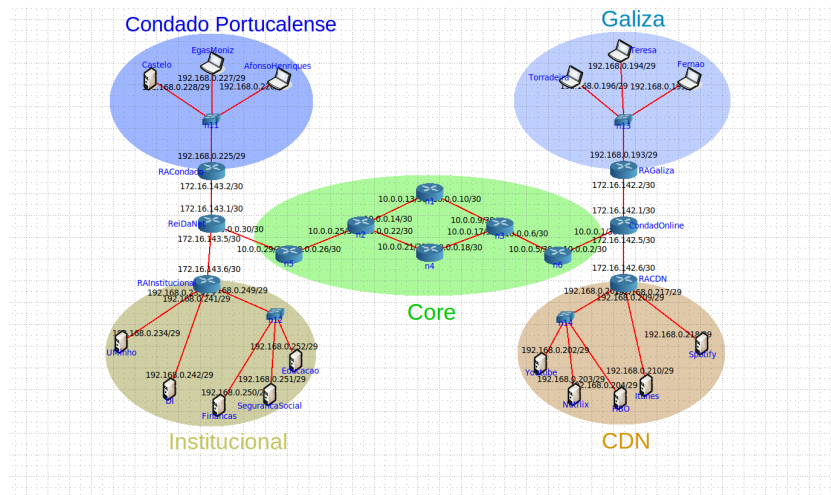


Figure 19: Topologia da Rede

3.1.1 Alínea a

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

Como podemos ver nas imagens que se seguem, o AfonsoHenriques tem conectividade com o servidor Financas e todos os servidores CDN.

```
vcmd
<core.33033/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=0.182 ms
64 bytes from 192.168.0.250: icmp_seq=2 ttl=61 time=0.092 ms
64 bytes from 192.168.0.250: icmp_seq=3 ttl=61 time=0.138 ms
64 bytes from 192.168.0.250: icmp_seq=4 ttl=61 time=0.130 ms
64 bytes from 192.168.0.250: icmp_seq=5 ttl=61 time=1.02 ms
64 bytes from 192.168.0.250: icmp_seq=6 ttl=61 time=0.124 ms
64 bytes from 192.168.0.250: icmp_seq=7 ttl=61 time=0.126 ms
64 bytes from 192.168.0.250: icmp_seq=8 ttl=61 time=0.109 ms
64 bytes from 192.168.0.250: icmp_seq=9 ttl=61 time=0.113 ms
64 bytes from 192.168.0.250: icmp_seq=10 ttl=61 time=0.134 ms
64 bytes from 192.168.0.250: icmp_seq=11 ttl=61 time=0.129 ms
^C
--- 192.168.0.250 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10201ms
rtt min/avg/max/mdev = 0.092/0.209/1.024/0.258 ms
root@AfonsoHenriques:/tmp/pycore.33033/AfonsoHenriques.conf#
```

Figure 20: Conectividade com o servidor Financas

```
<core.33033/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=0.174 ms
64 bytes from 192.168.0.204: icmp_seq=2 ttl=55 time=0.206 ms
64 bytes from 192.168.0.204: icmp_seq=3 ttl=55 time=0.147 ms
^C
--- 192.168.0.204 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2043ms
rtt min/avg/max/mdev = 0.147/0.175/0.206/0.024 ms
root@AfonsoHenriques:/tmp/pycore.33033/AfonsoHenriques.conf#
```

Figure 21: Conectividade com o servidor HBO

```
<core.33033/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=0.159 ms
64 bytes from 192.168.0.210: icmp_seq=2 ttl=55 time=0.203 ms
64 bytes from 192.168.0.210: icmp_seq=3 ttl=55 time=0.144 ms
^C
--- 192.168.0.210 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2028ms
rtt min/avg/max/mdev = 0.144/0.168/0.203/0.025 ms
root@AfonsoHenriques:/tmp/pycore.33033/AfonsoHenriques.conf#
```

Figure 22: Conectividade com o servidor Itunes

```
<core.33033/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=0.163 ms
64 bytes from 192.168.0.203: icmp_seq=2 ttl=55 time=0.218 ms
64 bytes from 192.168.0.203: icmp_seq=3 ttl=55 time=0.194 ms
^C
--- 192.168.0.203 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.163/0.191/0.218/0.022 ms
root@AfonsoHenriques:/tmp/pycore.33033/AfonsoHenriques.conf#
```

Figure 23: Conectividade com o servidor Netflix

```
<core.33033/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=0.161 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=55 time=0.182 ms
64 bytes from 192.168.0.218: icmp_seq=3 ttl=55 time=0.168 ms
^C
--- 192.168.0.218 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2046ms
rtt min/avg/max/mdev = 0.161/0.170/0.182/0.008 ms
root@AfonsoHenriques:/tmp/pycore.33033/AfonsoHenriques.conf#
```

Figure 24: Conectividade com o servidor Spotify

```
<core.33033/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=0.363 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.145 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.172 ms
^C
--- 192.168.0.202 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2046ms
rtt min/avg/max/mdev = 0.145/0.226/0.363/0.097 ms
root@AfonsoHenriques:/tmp/pycore.33033/AfonsoHenriques.conf#
```

Figure 25: Conectividade com o servidor Youtube

3.1.2 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*

Tanto o AfonsoHenriques como a Teresa têm apenas duas entradas nas respetivas tabelas de encaminhamento.

Analisando primeiro a tabela de encaminhamento do AfonsoHenriques.

```
root@AfonsoHenriques:/tmp/pycore.35445/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.35445/AfonsoHenriques.conf#
```

Figure 26: Tabela de encaminhamento no AfonsoHenriques

A primeira entrada, com o destino 0.0.0.0 e *gateway* 192.168.0.225, também conhecida como *default*, serve para enviar um pacote para fora do Condado Portucalense. Enquanto a outra entrada, com o destino 192.168.0.224 e *gateway* 0.0.0.0, que serve enviar para a própria rede (Condado Portucalense).

```
root@Teresa:/tmp/pycore.35445/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.35445/Teresa.conf#
```

Figure 27: Tabela de encaminhamento na Teresa

Tal como acontece com o AfonsoHenriques, a primeira entrada na tabela de encaminhamento da Teresa tem como destino 0.0.0.0 e *gateway* 192.168.0.192 que serve para enviar um pacote para fora da Galiza. Enquanto a outra tem como destino 192.168.0.192 e *gateway* 0.0.0.0, que serve enviar para a Galiza.

3.1.3 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.

Ao analisarmos o comportamento dos *routers* do core da rede, recorrendo ao *Wireshark*, conseguimos perceber que, o único *router* que, efetivamente, recebia pacotes era o *router* n5.

Os equipamentos que não permitem o encaminhamento correto dos pacotes são:

Router n5 Este *router* não tinha na sua tabela de encaminhamento nenhuma entrada com o destino do Condado Portucalense e, por isso, tivemos, então, que adicionar essa rota utilizando o comando ***ip route add 192.168.0.192/29 via 10.0.0.25***

Router n2 Este *router* tem a seguinte entrada na sua tabela de encaminhamento:

```
192.168.0.192 10.0.0.25 255.255.255.254 UG 0 0 0 eth2
```

Figure 28: Entrada na tabela de encaminhamento do *router* n2

No entanto, esta nesta entrada tanto o *gateway*('10.0.0.25') como a máscara('/31') estão erradas e, por isso, tivemos de o remover recorrendo ao comando ***ip route del 192.168.0.194/31***

Router n1 Este *router* tem a seguinte entrada na sua tabela de encaminhamento:

```
192.168.0.192 10.0.0.14 255.255.255.248 UG 0 0 0 eth1
```

Figure 29: Entrada na tabela de encaminhamento do *router* n1

Esta entrada está a voltar para o n2 em vez de avançar para o n3 (voltar para trás em vez de avançar) e, por essa razão precisamos de remover esta entrada da tabela de encaminhamento e adicionar a entrada no sentido correto. Para isso executamos os seguintes comandos, consecutivamente.

ip route del 192.168.0.192/29 via 10.0.0.14

ip route add 192.168.0.192/29 via 10.0.0.9

Router n3 Este *router* tem a seguinte entrada na sua tabela de encaminhamento:

```
192.168.0.192 10.0.0.18 255.255.255.240 UG 0 0 0 eth1
```

Figure 30: Entrada na tabela de encaminhamento do *router* n3

Nesta entrada tanto o sentido em que avança como a máscara estão incorretos, e, por isso, tivemos que remover esta entrada. Não tivemos que introduzir a entrada correta porque esta já constava na tabela.

Comando utilizado ***ip route del 192.168.0.192/28 via 10.0.0.18***

Apesar de, com a adição e remoção destas rotas o tráfego já chegar ao ISP Con-dadOnline, ainda não conseguimos estabelecer a ligação entre o AfonsoHenriques e a Teresa. Foi, por isso, preciso executar o comando ***ip route add 192.168.0.224/29 via 172.16.142.1*** no *router* RAGaliza.

```

vcmnd
<es,conf# traceroute 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.110 ms 0.008 ms 0.015 ms
 2 172.16.143.1 (172.16.143.1) 0.022 ms 0.006 ms 0.006 ms
 3 10.0.0.29 (10.0.0.29) 0.033 ms 0.009 ms 0.008 ms
 4 10.0.0.25 (10.0.0.25) 0.071 ms 0.011 ms 0.011 ms
 5 10.0.0.13 (10.0.0.13) 0.088 ms 0.034 ms 0.029 ms
 6 10.0.0.17 (10.0.0.17) 0.092 ms 0.036 ms 0.014 ms
 7 10.0.0.5 (10.0.0.5) 0.067 ms 0.018 ms 0.030 ms
 8 10.0.0.1 (10.0.0.1) 0.051 ms 0.019 ms 0.018 ms
 9 172.16.142.2 (172.16.142.2) 0.090 ms 0.029 ms 0.020 ms
10 192.168.0.194 (192.168.0.194) 0.055 ms 0.023 ms 0.022 ms
root@AfonsoHenriques:/tmp/pycore.45201/AfonsoHenriques,conf#

```

Figure 31: Comando *traceroute* no AfonsoHenriques para a Teresa

3.1.4 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.

- 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

As rotas escolhidas pela D.Teresa e pelo D.Afonso Henriques eram diferentes. No caminho entre um e o outro, a D.Teresa optava pelo n4, e o D.Afonso pelo n1. Contudo, no trajeto em direção à D.Teresa, a ligação 'saltava' do n1 para a interface do n3 que ligaria ao n4. Assim, alterou-se o caminho para passar pelo n4 em vez do n1, com os next-hops corretos.

6	7.761984822	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=8x001c, seq=1/256, ttl=55 (reply in 7)
7	7.761994738	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=8x001c, seq=1/256, ttl=64 (request in 6)
8	8.803194547	192.168.0.193	224.0.0.5	OSPF	78 Hello Packet	
9	8.781958149	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=8x001c, seq=2/512, ttl=55 (reply in 10)
10	8.781969222	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=8x001c, seq=2/512, ttl=64 (request in 9)
11	9.805884529	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) request	id=8x001c, seq=3/768, ttl=55 (reply in 12)
12	9.805897619	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) reply	id=8x001c, seq=3/768, ttl=64 (request in 11)

Figure 32: Comando *ping* do AfonsoHenriques para a Teresa

12	5.753887729	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) request	id=8x001b, seq=1/256, ttl=55 (reply in 13)
13	5.753892663	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) reply	id=8x001b, seq=1/256, ttl=64 (request in 12)
14	6.001069127	192.168.0.225	224.0.0.5	OSPF	78 Hello Packet	
15	6.772904479	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) request	id=8x001b, seq=2/512, ttl=55 (reply in 16)
16	6.772913931	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) reply	id=8x001b, seq=2/512, ttl=64 (request in 15)
17	7.674672395	fe80::a46c:a6ff:fe11:ff02::fd		NONE	203 Standard query 0x0000 PTR nfs_tcp-local, "Q" question PTR _ipp_tcp...	
18	7.797101284	192.168.0.194	192.168.0.226	ICMP	98 Echo (ping) request	id=8x001b, seq=3/768, ttl=55 (reply in 19)
19	7.797116039	192.168.0.226	192.168.0.194	ICMP	98 Echo (ping) reply	id=8x001b, seq=3/768, ttl=64 (request in 18)

Figure 33: Comando *ping* da Teresa para o AfonsoHenriques

- 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. Para fazer o maior número de *hops* possíveis para chegar ao ponto.

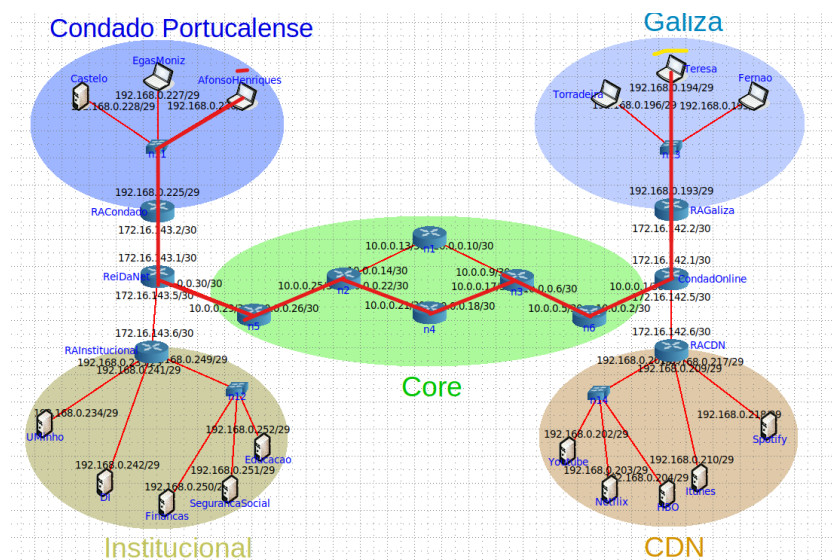


Figure 34: Rota seguida pelo Afonso

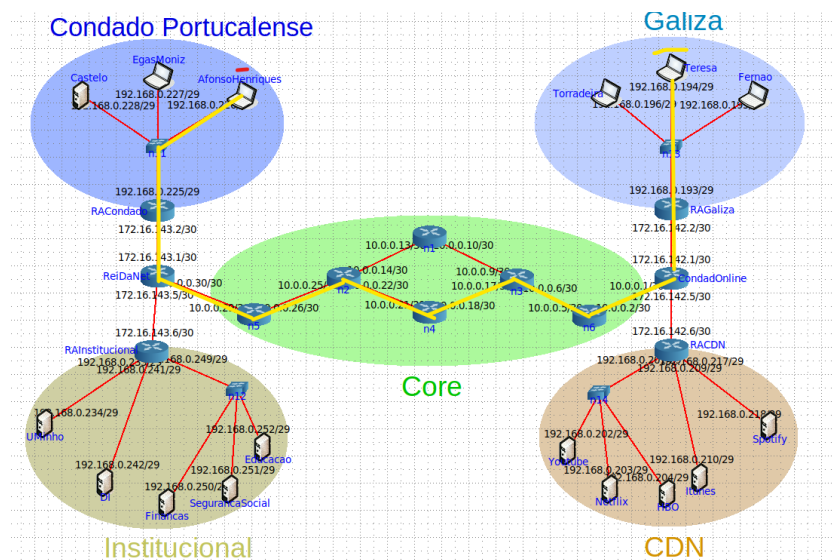


Figure 35: Rota seguida pela Teresa

As rotas são iguais, como é evidente nas Figuras 32 e 33.

3.1.5 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
```

Existe uma correspondência (*match*) nesta entrada para pacotes enviados para o polo Galiza? E para CDN? Caso seja essa a entrada utilizada para o encaminhamento, permitirá o funcionamento esperado do dispositivo?

Ofereça uma explicação pela qual essa entrada é ou não utilizada.

A entrada não é utilizada nem pela Galiza nem pelo CDN uma vez que a máscara da rota está errada tal como o next-hop é no sentido oposto. A máscara deveria ser /29 em vez de /28.

Caso a máscara e o sentido estivessem corretos só seria possível encaminhar para a Galiza, tendo em conta o destino.

3.1.6 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 endereços privados, uma vez que são endereços pertencentes ao bloco 172.16.0.0 - 172.31.255.255/12, bem como os endereços utilizados no core da rede que pertencem ao bloco 10.0.0.0 - 10.255.255.255/8. Estes dois blocos são endereços privados.

3.1.7 Alínea g

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

Como é possível constatar ao visualizar a topologia, os *switches* utilizados não têm nenhum endereço IP atribuído. Os *switches* podem, ou não, ter um IP atribuído e, serve apenas para acessar ao IP, uma vez que estes são transparentes, ou seja os *hosts* não tem capacidade de reconhecer a presença dos *switches*, para além disso os *switches* usam endereços MAC para gerenciar os datagramas.

3.2 Exercício 2

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

3.2.1 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. Explícite ainda a utilidade de uma rota *default*.

Para apagar a rota *default* do Castelo utilizamos o seguinte comando:

```
root@Castelo:/tmp/pycore.45201/Castelo.conf# ip route del default
```

Figure 36: Comando de remoção da rota *default*

Para que o Castelo continuasse a ter acesso a cada um dos três polos precisamos de executar os seguintes comandos no Castelo:

```
ip route add 192.168.0.232/29 via 192.168.0.225
```

```
ip route add 192.168.0.240/29 via 192.168.0.225
```

```
ip route add 192.168.0.248/29 via 192.168.0.225
```

```
ip route add 192.168.0.200/29 via 192.168.0.225
```

```
ip route add 192.168.0.208/29 via 192.168.0.225
```

```
ip route add 192.168.0.216/29 via 192.168.0.225
```

```
ip route add 192.168.0.192/29 via 192.168.0.225
```

A tabela de encaminhamento do Castelo resultante é a seguinte:

```
root@Castelo:/tmp/pycore.45201/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
```

Figure 37: Tabela de encaminhamento do Castelo

Para comprovar que a conectividade foi restabelecida seguem-se as seguintes imagens que mostram o resultado da execução do comando *ping* para os diferentes servidores das diferentes sub-redes dos 3 polos.

```
rtt min/avg/max/mdev = 0,086/0,134/0,183/0,048 ms
root@Castelo:/tmp/pycore.45201/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=0,200 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=55 time=0,224 ms
^C
--- 192.168.0.194 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1022ms
rtt min/avg/max/mdev = 0,200/0,212/0,224/0,012 ms
```

Figure 38: Conectividade com o *host* Teresa do polo Condado Portucalense

```
root@Castelo:/tmp/pycore.45201/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=0,172 ms
64 bytes from 192.168.0.242: icmp_seq=2 ttl=61 time=0,090 ms
^C
--- 192.168.0.242 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1030ms
rtt min/avg/max/mdev = 0,090/0,131/0,172/0,041 ms
```

Figure 39: Conectividade com o *host* DI do polo Intitucional

```
root@Castelo:/tmp/pycore.45201/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=0,103 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=61 time=0,110 ms
^C
--- 192.168.0.234 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1012ms
rtt min/avg/max/mdev = 0,103/0,106/0,110/0,003 ms
```

Figure 40: Conectividade com o *host* UMinho do polo Intitucional

```
root@Castelo:/tmp/pycore.45201/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=0,174 ms
64 bytes from 192.168.0.251: icmp_seq=2 ttl=61 time=0,111 ms
^C
--- 192.168.0.251 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0,111/0,142/0,174/0,031 ms
```

Figure 41: Conectividade com o *host* SegurancaSocial do polo Intitucional

```
root@Castelo:/tmp/pycore.45201/Castelo.conf# ping 192.168.0.202
PING 192.168.0.202 (192.168.0.202) 56(84) bytes of data:
64 bytes from 192.168.0.202: icmp_seq=1 ttl=55 time=0,241 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0,162 ms
^C
--- 192.168.0.202 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1028ms
rtt min/avg/max/mdev = 0,162/0,201/0,241/0,039 ms
```

Figure 42: Conectividade com o *host* Youtube do polo CDN

```

root@Castelo:/tmp/pycore.45201/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=0.153 ms
64 bytes from 192.168.0.218: icmp_seq=2 ttl=55 time=0.219 ms
^C
--- 192.168.0.218 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.153/0.186/0.219/0.033 ms

```

Figure 43: Conectividade com o *host* Spotify do polo CDN

```

root@Castelo:/tmp/pycore.45201/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=0.247 ms
64 bytes from 192.168.0.210: icmp_seq=2 ttl=55 time=0.192 ms
^C
--- 192.168.0.210 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1024ms
rtt min/avg/max/mdev = 0.192/0.219/0.247/0.027 ms

```

Figure 44: Conectividade com o *host* Itunes do polo CDN

Como é fácil perceber, ao removermos a rota *default* da tabela de encaminhamento do Castelo, tivemos que adicionar, manualmente, as rotas para os diferentes polos e as diferentes redes. Podemos ainda realçar que, a tabela de encaminhamento do Castelo que antes tinha 2 entradas agora tem 8 entradas e, caso seja acrescentado um novo polo à topologia terá de ser criada uma nova rota, manualmente, para que o Castelo consiga aceder a esse polo, o que não teria de acontecer com a rota default.

3.2.2 Alínea b

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

Atendendo a que o nosso grupo é o 55, o endereço atribuído pelo ISP à rede será 172.16.55.128/26

Como o endereço do ISP tem /26 como máscara de rede, sobram 6 bits para distribuir para as sub-redes e para os *hosts*.

Como nos é pedido que seja possível estabelecer pelo menos 3 sub-redes, optamos por reservar 2 *bits* para a identificação das sub-redes, sobrando, assim, 4 *bits* para a os *hosts*. De salientar que nos era pedido para que fosse possível ter 10 ou mais *hosts* em cada sub-rede, o que é possível como o nosso esquema de endereçamento.

Sub-rede	Rede Gerada	Intervalos de Interface Possíveis
00	172.16.55.128	192.168.040.129 a 192.040.142
01	192.168.040.144	192.168.040.145 a 192.040.158
10	192.168.040.160	192.168.040.161 a 192.040.174
11	192.168.040.176	192.168.040.177 a 192.040.190

Table 1: Endereços de cada sub-rede e *hosts*

Na imagem seguinte é possível visualizar a topologia atualizada com o polo que acrescentamos.

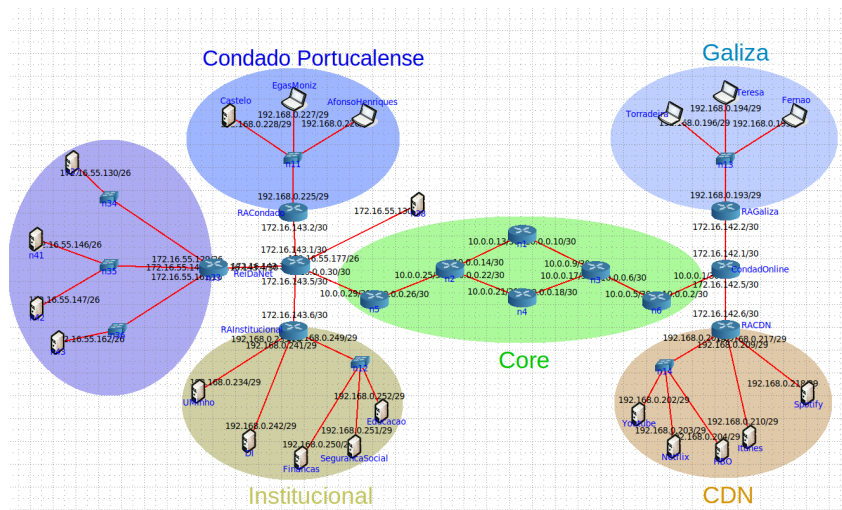


Figure 45: Topologia alterada

3.2.3 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.

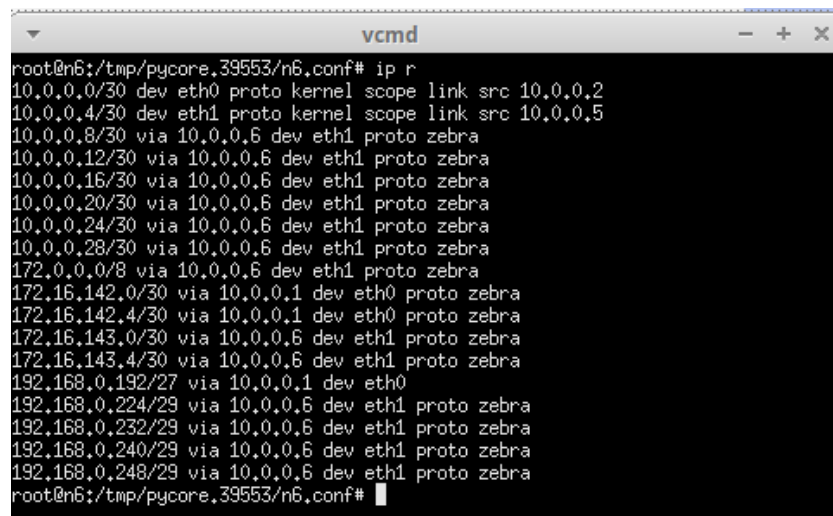
Existe algum *host* com o qual não seja possível comunicar? Porquê?

3.3 Exercício 3

Ao planejar 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.

3.3.1 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



```
root@n6:/tmp/pycore.39553/n6.conf# ip r
10.0.0.0/30 dev eth0 proto kernel scope link src 10.0.0.2
10.0.0.4/30 dev eth1 proto kernel scope link src 10.0.0.5
10.0.0.8/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.12/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.16/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.20/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.24/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.28/30 via 10.0.0.6 dev eth1 proto zebra
172.0.0.0/8 via 10.0.0.6 dev eth1 proto zebra
172.16.142.0/30 via 10.0.0.1 dev eth0 proto zebra
172.16.142.4/30 via 10.0.0.1 dev eth0 proto zebra
172.16.143.0/30 via 10.0.0.6 dev eth1 proto zebra
172.16.143.4/30 via 10.0.0.6 dev eth1 proto zebra
192.168.0.192/27 via 10.0.0.1 dev eth0
192.168.0.224/29 via 10.0.0.6 dev eth1 proto zebra
192.168.0.232/29 via 10.0.0.6 dev eth1 proto zebra
192.168.0.240/29 via 10.0.0.6 dev eth1 proto zebra
192.168.0.248/29 via 10.0.0.6 dev eth1 proto zebra
root@n6:/tmp/pycore.39553/n6.conf#
```

Figure 46: Tabela de roteamento do n6 com Galiza e CDN

3.3.2 Alínea b

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



```
root@n6:/tmp/pycore.39553/n6.conf# ip r
10.0.0.0/30 dev eth0 proto kernel scope link src 10.0.0.2
10.0.0.4/30 dev eth1 proto kernel scope link src 10.0.0.5
10.0.0.8/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.12/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.16/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.20/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.24/30 via 10.0.0.6 dev eth1 proto zebra
10.0.0.28/30 via 10.0.0.6 dev eth1 proto zebra
172.0.0.0/8 via 10.0.0.6 dev eth1 proto zebra
172.16.142.0/30 via 10.0.0.1 dev eth0 proto zebra
172.16.142.4/30 via 10.0.0.1 dev eth0 proto zebra
172.16.143.0/30 via 10.0.0.6 dev eth1 proto zebra
172.16.143.4/30 via 10.0.0.6 dev eth1 proto zebra
192.168.0.192/27 via 10.0.0.1 dev eth0
192.168.0.224/27 via 10.0.0.6 dev eth1
root@n6:/tmp/pycore.39553/n6.conf#
```

Figure 47: Tabela de roteamento do n6 com CondadoPortucalense e Institucional

/cidr	/26	/27	/28	/29
			192.168.0.192	192.168.0.192
				192.168.0.200
			192.168.0.208	192.168.0.208
				192.168.0.216
		192.168.0.224	192.168.0.224	192.168.0.224
				192.168.0.232
			192.168.0.240	192.168.0.240
				192.168.0.248
group size	64	32	16	8
subnet mask	192	224	240	248

Figure 48: Esquema do supernetting feito

3.3.3 Alínea c

Comente os aspetos positivos e negativos do uso do Supernetting.

O supernetting, por um lado, é capaz de reduzir o tráfego da rede, aumentar a velocidade da procura em tabelas de roteamento, e otimizar o tamanho da tabela de endereçamento de um router, pois condensa várias entradas numa só. Por outro lado, estas vantagens requerem que todas as redes da super-net usem a mesma classe de endereço de IP, e portanto condicionam toda a rede a estar na mesma classe. Apresentando-se como o inverso do subnetting, não irá diminuir a escassez de endereços, que é o objetivo do seu processo 'oposto'. Analisando as vantagens e desvantagens, o 'saldo' do supernetting é bastante positivo.

4 Conclusão

Na primeira fase deste trabalho abordamos a transmissão de dados referente a máquinas dentro da mesma rede e a necessidade, ou não, de fragmentação no envio de pacotes de dados.

Na segunda fase abordamos o funcionamento do encaminhamento e endereçamento entre os vários polos distintos, cada um com a sua sub-rede e a diferença entre os dois tipos de encaminhamento e formas de endereçar redes.

Tivemos como dificuldade o exercício 2, alínea c) da segunda parte do trabalho, pois no endereçamento atribuímos o IP do novo *host*, mas não conseguimos conectar este *host* e não entendemos ao certo o motivo de esta conexão não ser realizada.

No entanto, este trabalho foi bastante importante para consolidar e colocar em prática os conhecimentos adquiridos nas aulas teóricas.