

Redes de Computadores

Trabalho Prático 2

Pedro Afonso Moreira Lopes [A100759], Gonalo Machado Daniel Costa [A100824] e Jos  Eduardo Silva Monteiro Santos Oliveira [A100547]

Parte 1 - Datagramas IP e Fragmenta o

1.1 Prepare uma topologia CORE para verificar o comportamento do traceroute. Na topologia deve existir: um host (pc) cliente designado Lost, cujo router de acesso   RA1; o router RA1 est  simultaneamente ligado a dois routers no core da rede RC1 e RC2; estes est o conectados a um router de acesso RA2, que por sua vez, se liga a um host (servidor) designado Found. Ajuste o nome dos equipamentos atribu dos por defeito para o enunciado. Apenas nas liga es (links) da rede de core, estabelea um tempo de propaga o de 15 ms. Ap s ativar a topologia, note que pode n o existir conectividade IP imediata entre Lost e Found at  que o an ncio de rotas entre routers estabilize.

a) Active o Wireshark no host Lost. Numa shell de Lost execute o comando traceroute -I para o endereo IP do Found. Registe e analise o tr fego ICMP enviado pelo sistema Lost e o tr fego ICMP recebido como resposta. Explique os resultados obtidos tendo em conta o princ pio de funcionamento do traceroute.

A primeira coisa a fazer nesta quest o foi construir a topologia CORE. Assim sendo, definimos primeiro o Host (pc) como Lost, que vai estar conectado ao router RA1. Por sua vez, este est  conectado aos routers RC1 e RC2, que est o conectados ao router RA2. Finalmente, este  ltimo router est  conectado ao Host (servidor) chamado de Found, tal como se pode ver na seguinte imagem.

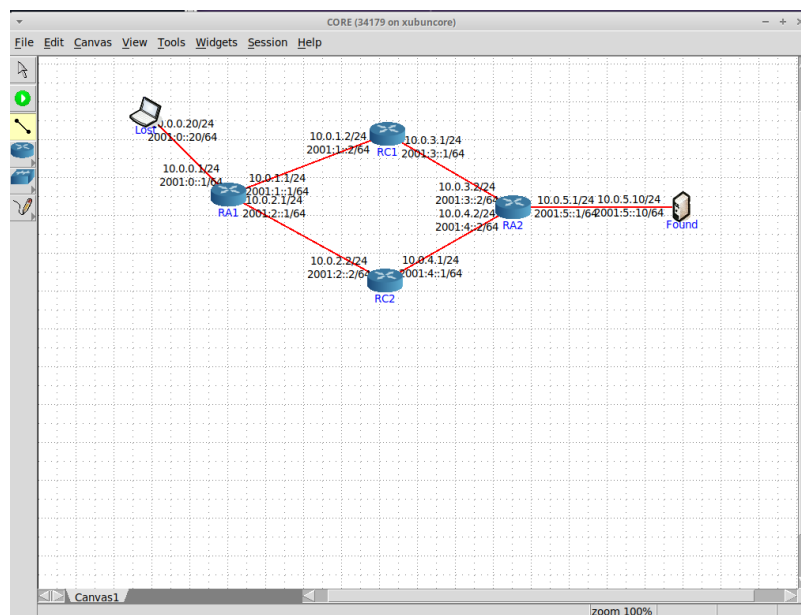


Fig.1. Topologia CORE

Após a configuração desta topologia, ativamos então a captura de tráfego com o Wireshark no host Lost. Finalmente, executamos o comando *traceroute -I 10.0.5.10* (sendo este o endereço IP do host Found), que nos deu os seguintes resultados:

```
root@Lost:/tmp/pycore.33551/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) 0.051 ms 0.012 ms 0.059 ms
 2 10.0.1.2 (10.0.1.2) 30.995 ms 30.978 ms 30.969 ms
 3 10.0.3.2 (10.0.3.2) 61.358 ms 61.354 ms 61.348 ms
 4 10.0.5.10 (10.0.5.10) 61.342 ms 61.337 ms 61.330 ms
root@Lost:/tmp/pycore.33551/Lost.conf#
```

Fig.2. Resultados do comando *traceroute -I 10.0.5.10* no terminal

Na seguinte imagem, conseguimos ver o tráfego ICMP capturado na shell do host Lost:

No.	Time	Source	Destination	Protocol	Length	Info
57	68.808117559	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response...
58	68.808142899	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
59	68.808158397	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=2/512, ttl=1 (no response...
60	68.808167689	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
61	68.808175488	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response...
62	68.808182455	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
63	68.808235833	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no respons...
64	68.808272822	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no respons...
65	68.808281493	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no respons...
66	68.808288991	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no respons...
67	68.808295995	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no respons...
68	68.808301481	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no respons...
69	68.808309989	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in ...
70	68.808316557	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in ...
71	68.808323283	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (reply in ...
72	68.808331363	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (reply in ...
73	68.808338640	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (reply in ...
74	68.808344334	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (reply in ...
75	68.808351651	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (reply in ...
76	68.810631171	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=17/4352, ttl=6 (reply in ...
77	68.810650959	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=18/4608, ttl=6 (reply in ...
78	68.810661314	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=19/4864, ttl=7 (reply in ...
79	68.869081714	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
80	68.869086996	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
81	68.869088730	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
82	68.871324445	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=20/5120, ttl=7 (reply in ...
83	68.871349064	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=21/5376, ttl=7 (reply in ...
84	68.871360692	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001b, seq=22/5632, ttl=8 (reply in ...
85	68.899885965	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
86	68.899891148	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
87	68.899893002	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
88	68.899894316	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=10/2560, ttl=61 (request ...
89	68.899895729	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=11/2816, ttl=61 (request ...
90	68.899897092	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=12/3072, ttl=61 (request ...
91	68.899898325	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=13/3328, ttl=61 (request ...
92	68.899899648	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=14/3584, ttl=61 (request ...
93	68.899901012	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=15/3840, ttl=61 (request ...
94	68.899902255	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=16/4096, ttl=61 (request ...
95	68.899903538	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=17/4352, ttl=61 (request ...
96	68.899904941	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=18/4608, ttl=61 (request ...
97	68.899906254	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=19/4864, ttl=61 (request ...
98	68.932930914	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=20/5120, ttl=61 (request ...
99	68.932935836	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=21/5376, ttl=61 (request ...
100	68.932937610	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001b, seq=22/5632, ttl=61 (request ...

Fig.3. Tráfego ICMP capturado depois da execução do comando *traceroute*

Aqui nesta imagem, como podemos ver, entre todos os datagramas capturados, temos alguns apresentados a preto, pois estes são os datagramas de erro, ou seja, aqueles que são criados para devolver uma mensagem de erro para o router origem, avisando-o que o TTL (Time-To-Live) foi excedido. Com isto, o comando *traceroute* vai mandar conjuntos de 3 datagramas com o TTL a crescer desde 1 até à quantidade necessária para chegar ao destino. Assim, é possível “descobrir” o caminho que o datagrama percorreu. Sendo assim, podemos concluir que estes foram os resultados esperados.

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 4. Através da figura 3 em cima, se o valor do campo TTL for inferior a 4, o pacote será descartado pelos 3 routers e será enviada a mensagem erro ICMP a informar a origem que o Time-To-Live foi excedido em trânsito. Por outro lado, caso seja 4 ou superior, o pacote chegará ao destino e haverá uma resposta.

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.

De maneira a obter o valor médio do tempo de ida-e-volta (RTT - Round-Trip Time), necessitamos de verificar o output do comando *traceroute* para o destino Found na figura 2, que vai nos indicar os 3 tempos obtidos para cada salto. Com isto, o cálculo do valor médio do tempo será então:

$$RTT = (61.342 + 61.337 + 61.330) / 3 = 61.336(3)$$

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 *One-Way Delay* não irá levar a um cálculo com precisão ao dividir o RTT por dois pois, antes de tudo, o RTT é a soma dos atrasos de ida-e-volta de uma mensagem entre 2 pontos, enquanto que o *One-Way Delay* é a soma dos atrasos de apenas um sentido. Além disto, este cálculo não será fácil numa rede real, graças a algumas limitações, como por exemplo:

- O requisito de cooperação intensiva entre ambos os computadores para o cálculo ser preciso pois a mínima imprecisão nas medições pode levar a erros significantes.
- A variação do atraso, pois este pode ser afetado por vários fatores, como o congestionamento da rede, erros de transmissão, roteamento de pacotes, entre outros.
- Um número grande de saltos, sendo que cada um destes saltos pode afetar negativamente o cálculo de forma imprevisível.

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

Documente e justifique todas as respostas às seguintes alíneas:

Antes de tudo, começamos por executar o comando *traceroute -I marco.uminho.pt* para a resolução das alíneas deste exercício.

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

Para descobrir o endereço IP da interface ativa do nosso computador, localizamos a primeira mensagem ICMP capturada e verificamos nele o IP fonte, que foi 172.26.41.9, como se pode ver na figura abaixo.

```
> Frame 168: 76 bytes on wire (608 bits), 76 bytes captured (608 bits) on interface any, id 0
> Linux cooked capture v1
> Internet Protocol Version 4, Src: 172.26.41.9, Dst: 193.136.9.240
  - 0100 .... = Version: 4
  - .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  - Total Length: 60
  - Identification: 0xb6f4 (46836)
  > 000. .... = Flags: 0x0
  - ...0 0000 0000 0000 = Fragment Offset: 0
  > Time to Live: 1
  - Protocol: ICMP (1)
  - Header Checksum: 0x6231 [validation disabled]
  - [Header checksum status: Unverified]
  - Source Address: 172.26.41.9
  - Destination Address: 193.136.9.240
> Internet Control Message Protocol
```

Fig.4. Informação relativa à primeira mensagem ICMP capturada

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

O protocolo é ICMP : *Internet Control Message Protocol*, que é identificado pelo valor 1 na camada IP, como se pode ver na figura 4.

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 tamanho do cabeçalho IPv4 é de 20 bytes. O tamanho do payload é calculado a partir da diferença do tamanho total do pacote subtraindo o tamanho do cabeçalho, ou seja, será então:

$$60 - 20 = 40 \text{ bytes}$$

d) O datagrama IP foi fragmentado? Justifique.

Não. Como o Fragment Offset = 0, isto significa que estamos no início do pacote e, como o campo “More Fragments” também não está definido, isto significa que esta é a última parte do pacote. Assim sendo, o datagrama IP não foi fragmentado pois o seu tamanho não excedia o valor do MTU.

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.

No.	Time	Source	Destination	Protocol	Length	Info
191	12.020459750	172.16.115.252	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
192	12.021515749	172.16.115.252	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
195	12.021519590	172.16.115.252	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
184	12.018824977	172.16.2.1	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
186	12.018953345	172.16.2.1	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
188	12.019503275	172.16.2.1	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
197	12.024877694	172.26.254.254	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
198	12.024879091	172.26.254.254	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
199	12.025143789	172.26.254.254	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
168	12.017400776	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=1/256, ttl=1 (no response found!)
169	12.017417398	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=2/512, ttl=1 (no response found!)
170	12.017422706	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=3/768, ttl=1 (no response found!)
171	12.017428852	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=4/1024, ttl=2 (no response found!)
172	12.017433601	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=5/1280, ttl=2 (no response found!)
173	12.017440236	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=6/1536, ttl=2 (no response found!)
174	12.017446452	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=7/1792, ttl=3 (no response found!)
175	12.017452807	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=8/2048, ttl=3 (no response found!)
176	12.017458465	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=9/2304, ttl=3 (no response found!)
177	12.017464122	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=10/2560, ttl=4 (reply in 190)
178	12.017469849	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=11/2816, ttl=4 (no response found!)
179	12.017475576	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=12/3072, ttl=4 (no response found!)
180	12.017481303	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=13/3328, ttl=5 (no response found!)
181	12.017485703	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=14/3584, ttl=5 (no response found!)
182	12.017490871	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=15/3840, ttl=5 (no response found!)
183	12.017497017	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=16/4096, ttl=6 (no response found!)
185	12.018933371	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=17/4352, ttl=6 (reply in 193)
187	12.018962495	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=18/4608, ttl=6 (reply in 194)
189	12.019617605	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=19/4864, ttl=7 (reply in 196)
190	12.019730329	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=10/2560, ttl=61 (request in 177)
193	12.021517215	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=17/4352, ttl=61 (request in 185)
194	12.021518123	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=18/4608, ttl=61 (request in 187)
196	12.021520498	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=19/4864, ttl=61 (request in 189)

Fig.5. Pacotes ICMP capturados, ordenados de acordo com o endereço IP

Analisando a sequência de mensagens ICMP, os únicos campos do cabeçalho IP que variam de pacote para pacote são:

- ID único dos pacotes.
- O header checksum.
- O TTL.

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

Os campos TTL aumentam 1 a cada 3 mensagens, como se pode ver na figura 7 e, se a Source for a mesma, os campos *Identification* são sequenciais.

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

No.	Time	Source	Destination	Protocol	Length	Info
184	12.018824977	172.16.2.1	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
186	12.018953345	172.16.2.1	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
188	12.019503275	172.16.2.1	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
190	12.019730329	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=10/2560, ttl=61 (request in 177)
191	12.020459750	172.16.115.252	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
192	12.021515749	172.16.115.252	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
193	12.021517215	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=17/4352, ttl=61 (request in 185)
194	12.021518123	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=18/4608, ttl=61 (request in 187)
195	12.021519599	172.16.115.252	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
196	12.021520498	193.136.9.240	172.26.41.9	ICMP	76	Echo (ping) reply id=0x0005, seq=19/4864, ttl=61 (request in 189)
197	12.024877694	172.26.254.254	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
198	12.024879091	172.26.254.254	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
199	12.025143789	172.26.254.254	172.26.41.9	ICMP	72	Time-to-live exceeded (Time to live exceeded in transit)
168	12.017490776	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=1/256, ttl=1 (no response found!)
169	12.017417398	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=2/512, ttl=1 (no response found!)
170	12.017422706	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=3/768, ttl=1 (no response found!)
171	12.017428852	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=4/1024, ttl=2 (no response found!)
172	12.017433601	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=5/1280, ttl=2 (no response found!)
173	12.017440236	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=6/1536, ttl=2 (no response found!)
174	12.017446452	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=7/1792, ttl=3 (no response found!)
175	12.017452807	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=8/2048, ttl=3 (no response found!)
176	12.017458465	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=9/2304, ttl=3 (no response found!)
177	12.017464122	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=10/2560, ttl=4 (reply in 190)
178	12.017469849	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=11/2816, ttl=4 (no response found!)
179	12.017475576	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=12/3072, ttl=4 (no response found!)
180	12.017481303	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=13/3328, ttl=5 (no response found!)
181	12.017485703	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=14/3584, ttl=5 (no response found!)
182	12.017490871	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=15/3840, ttl=5 (no response found!)
183	12.017497017	172.26.41.9	193.136.9.240	ICMP	76	Echo (ping) request id=0x0005, seq=16/4096, ttl=6 (no response found!)

Fig.7. Série de respostas ICMP TTL *Exceeded* no tráfego capturado ordenado por endereço destino

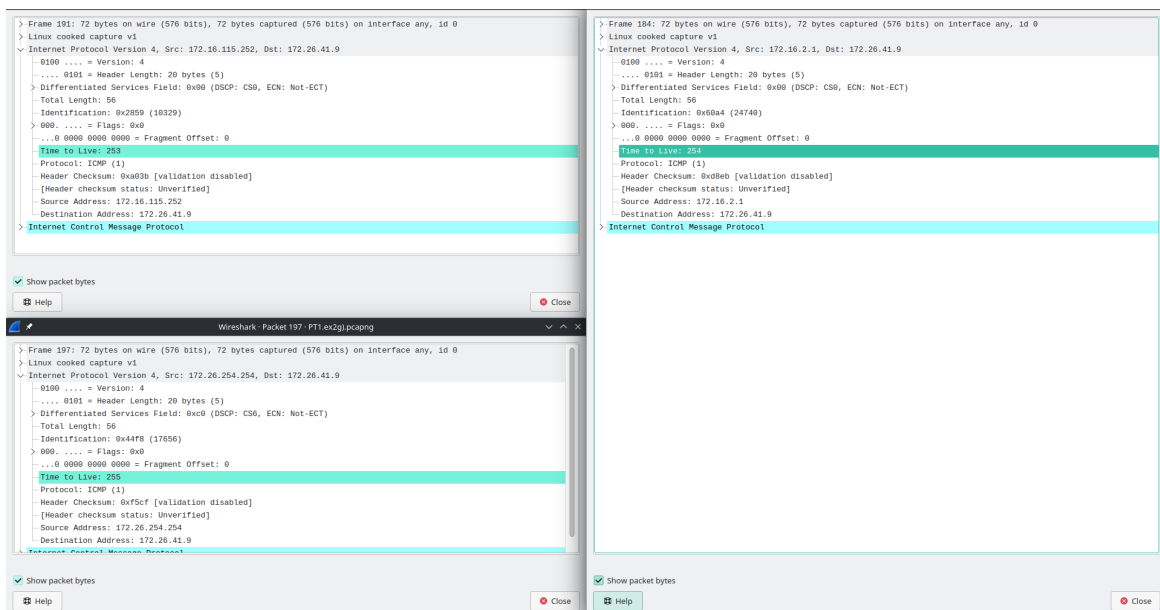


Fig.8. Informações relativas a alguns dos pacotes capturados

g.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?

O valor TTL não permanece constante e vai variar entre 253, 254 e 255, sendo cada um destes respetivos de uma source. Isto acontece pois são enviados 3 datagramas com um

certo TTL de cada vez e, sempre que algum conjunto de datagramas não é recebido, outros 3 datagramas são enviados de uma source diferente, tal como se pode ver na figura 8.

g.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 sempre enviadas na origem com um valor TTL relativamente alto pois, de forma a que a mensagem chegue à origem do pacote, tenta-se garantir que isto aconteça se a mensagem de resposta for enviada com um valor TTL relativamente alto, em vez do valor TTL do pacote.

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?

Sim, a informação contida no cabeçalho ICMP poderia ser incluída no cabeçalho IPv4 e isto teria vantagens a nível de espaço e velocidade de processamento, como por exemplo o processamento de pacotes mais rápido e mais eficiente.

Porém, teria também desvantagens pois o cabeçalho será mais complexo e terá maior tamanho, tendo assim menos espaço para a informação.

1.3 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”. Nota: Como alternativa para geração do tráfego pode usar o comando ping <opção> <bytes> marco.uminho.pt, onde a opção -l (Windows) ou -s (Linux, Mac) permite definir o número de bytes enviados no campo de dados do pacote ICMP. Documente e justifique todas as respostas às seguintes alíneas:

Como se pode ver pela figura seguinte, a primeira mensagem ICMP encontra-se identificada pelo número 100.

No.	Time	Source	Destination	Protocol	Length	Info
100	2.670880358	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request id=0x0008, seq=1/256, ttl=64 (reply in 103)
101	2.670905990	172.26.41.9	193.136.9.240	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=bb86)
102	2.670909343	172.26.41.9	193.136.9.240	IPv4	598	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=bb86)
103	2.674756558	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply id=0x0008, seq=1/256, ttl=61 (request in 100)
104	2.674757955	193.136.9.240	172.26.41.9	IPv4	598	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=19dc)
105	2.674759422	193.136.9.240	172.26.41.9	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=19dc)
109	3.672925977	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request id=0x0008, seq=2/512, ttl=64 (reply in 112)
110	3.672956009	172.26.41.9	193.136.9.240	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=bcb2)
111	3.672961387	172.26.41.9	193.136.9.240	IPv4	598	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=bcb2)
112	3.675499355	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply id=0x0008, seq=2/512, ttl=61 (request in 109)
113	3.675500822	193.136.9.240	172.26.41.9	IPv4	598	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=1a16)
114	3.675724003	193.136.9.240	172.26.41.9	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=1a16)
120	4.674482986	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request id=0x0008, seq=3/768, ttl=64 (reply in 123)
121	4.674513506	172.26.41.9	193.136.9.240	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=bd91)
122	4.674518745	172.26.41.9	193.136.9.240	IPv4	598	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=bd91)
123	4.678660761	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply id=0x0008, seq=3/768, ttl=61 (request in 120)
124	4.678741358	193.136.9.240	172.26.41.9	IPv4	598	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=1ae7)
125	4.679037276	193.136.9.240	172.26.41.9	IPv4	1516	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=1ae7)

Fig.9. Captura de tráfego gerado pelo comando *traceroute* com pacotes de tamanho 3514

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

Houve necessidade de fragmentar o primeiro pacote pois o tamanho deste excedia o MTU ou Maximum Transmission Unit (Tamanho máximo que um datagrama poderá ter, de modo a ser enviado pela rede). Como o tamanho dos pacotes enviados é igual a 3514 bytes, que é maior que o valor típico do MTU de 1500, vai ser necessário então necessário fragmentá-lo 3 vezes teoricamente, tendo em conta também o cabeçalho do protocolo IPv4 que ocupa 20 bytes.

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?

Como se pode ver na figura seguinte, temos aqui a impressão do primeiro fragmento do datagrama IP original.

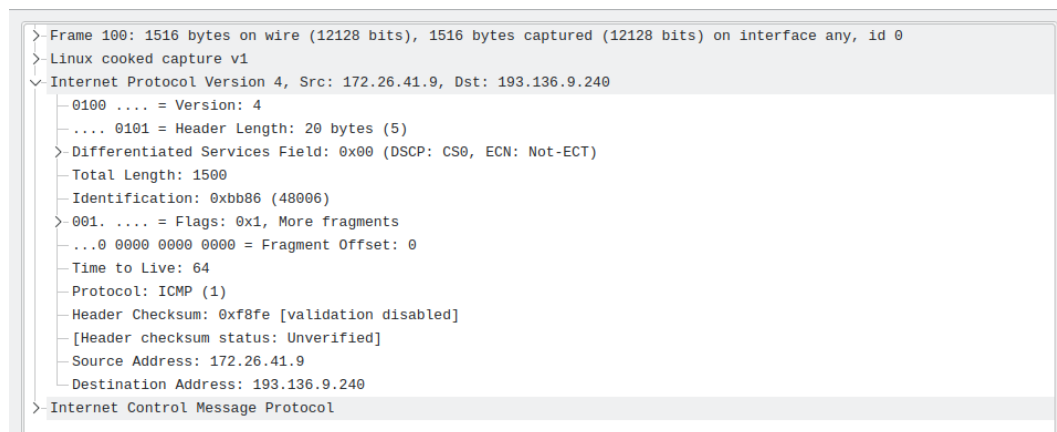


Fig.10. Informações relativas ao primeiro fragmento

Com estas informações acima, conseguimos indicar que o datagrama foi fragmentado e que este é o primeiro fragmento pois a flag “More fragments” está marcada a verdadeira e o Fragment Offset encontra-se igual a zero.

Relativamente ao tamanho do datagrama, este é de 1500 bytes.

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?

De seguida, conseguimos ver a impressão do segundo fragmento do datagrama IP original.


```
> Frame 101: 1516 bytes on wire (12128 bits), 1516 bytes captured (12128 bits) on interface any, id 0
> Linux cooked capture v1
  - Internet Protocol Version 4, Src: 172.26.41.9, 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: 0xbb86 (48006)
    > 001. .... = Flags: 0x1, More fragments
    - ...0 0000 1011 1001 = Fragment Offset: 1480
    - Time to Live: 64
    - Protocol: ICMP (1)
    - Header Checksum: 0xf845 [validation disabled]
    - [Header checksum status: Unverified]
    - Source Address: 172.26.41.9
    - Destination Address: 193.136.9.240
  > Data (1480 bytes)
```

Fig.11. Informações relativas ao segundo fragmento

Com estas informações recolhidas, conseguimos indicar que este não é o primeiro fragmento pois o Fragment Offset é diferente de zero. Existem também mais fragmentos pois a flag More fragments encontra-se assinalada com 1.

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, de acordo com os cálculos feitos, o número de fragmentos deve ser 3 pois, em 3514, cabem 2 fragmentos de offset 1480 e mais um de offset 554, como se pode ver pela conta $3514 - 2 * 1480 = 554$. Com isto, sabemos também que o último fragmento, teoricamente, transporta 554 bytes.

Porém, verificando o valor de bytes transportados no último fragmento fornecido pelo wireshark, como se pode ver na figura 12, descobrimos então que este valor é igual a 562, sendo este maior que o valor teórico calculado por 8 bytes.

```
Wireshark · Packet 102 · PT1-ex3.pcapng
> Frame 102: 598 bytes on wire (4784 bits), 598 bytes captured (4784 bits) on interface any, id 0
> Linux cooked capture v1
  - Internet Protocol Version 4, Src: 172.26.41.9, 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: 582
    - Identification: 0xbb86 (48006)
    > 000. .... = Flags: 0x0
    - ...0 0001 0111 0010 = Fragment Offset: 2960
    - Time to Live: 64
    - Protocol: ICMP (1)
    - Header Checksum: 0x1b23 [validation disabled]
    - [Header checksum status: Unverified]
    - Source Address: 172.26.41.9
    - Destination Address: 193.136.9.240
  > Data (562 bytes)
```

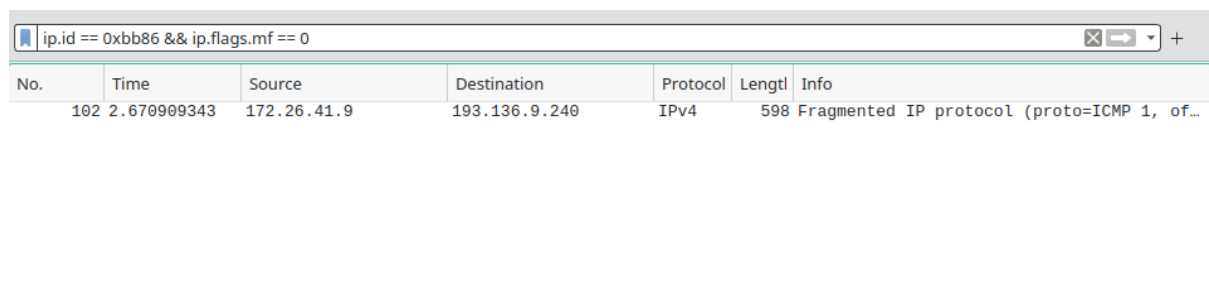
Fig.12. Informações relativas ao último fragmento

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 correspondente ao datagrama original pode ser detetado através da verificação da flag More Fragments e do ID do fragmento. Para tal então, a flag More Fragments tem de ser igual a zero pois não podem haver mais fragmentos a seguir do último e temos que marcar também o ID do datagrama original, para assim obtermos apenas fragmentos desse em específico.

Com isto, chegamos então ao último fragmento, sendo isto equivalente ao seguinte filtro no Wireshark:

ip.flags.mf == 0 && ip.id == 0xbb86



The image shows a Wireshark packet capture window with a filter applied: `ip.id == 0xbb86 && ip.flags.mf == 0`. The packet list shows a single entry:

No.	Time	Source	Destination	Protocol	Length	Info
102	2.670909343	172.26.41.9	193.136.9.240	IPv4	598	Fragmented IP protocol (proto=ICMP 1, of...

Fig.13. Captura do último fragmento correspondente ao datagrama original, como se pode ver pela figura 12 também

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 equipamento onde o datagrama IP original é reconstruído a partir dos fragmentos está marcado como destino, ou seja, no *193.136.9.240*, como se pode ver na figura 10. A reconstrução do datagrama IP original é feita no destino final pois, para isto ser possível, serão necessários todos os fragmentos criados para reconstruir a mensagem original e apenas.

Sabendo isto, a reconstrução só pode ser feita apenas no destino final e não em mais nenhum equipamento diferente.

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 variam no cabeçalho IP entre os diferentes fragmentos são o Fragment Offset, que permite saber o número de bytes que precedem este fragmento, e a flag More fragments, que indica se há mais fragmentos a seguir do atual.

Depois de todos os fragmentos chegarem ao destino, estes são ordenados através do Fragment Offset. Como por exemplo, o pacote com offset 0 vai ser o primeiro pois, tal como mencionado anteriormente, nenhum pacote o vai preceder. Sabendo isto, o pacote com

offset de 1480 vai ser então o segundo e, finalmente, temos o último que, ou vai ser identificado graças à flag More fragments estar igual a zero ou por ser o fragmento com o maior Fragment Offset.

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

Apenas o primeiro fragmento é identificado como um pacote ICMP pois, como todos os fragmentos têm a mesma identificação do datagrama original e numa questão de otimização, apenas o primeiro fragmento contém o cabeçalho ICMP completo e o resto dos fragmentos apenas têm a parte dos dados do datagrama original, sendo assim identificados como IPv4.

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 tamanho do datagrama a fim de se determinar se este deve se fragmentado é comparado com o valor do MTU (Maximum Transmission Unit). Neste caso, o MTU é 1500 bytes.

Caso se aumente este valor, vai haver menos fragmentação e menos fragmentos na rede, já que o valor necessário a atingir para ocorrer a fragmentação vai ser maior, levando a menos fragmentos a serem transmitidos e a demorarem mais tempo na transmissão, logo em caso de erro, o reenvio será mais demorado também.

Caso se diminua este valor, pelo contrário, haverá mais fragmentação e mais fragmentos na rede. Com isto, será possível também que os datagramas menores sejam transmitidos mais rapidamente.

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.

No.	Time	Source	Destination	Protocol	Length	Info
382	6.659952955	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request 1d=0x0003, seq=1/256, ttl=64 (reply in 384)
384	6.661982813	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply 1d=0x0003, seq=1/256, ttl=64 (request in 382)
385	7.661482559	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request 1d=0x0003, seq=2/512, ttl=64 (reply in 386)
386	7.664828371	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply 1d=0x0003, seq=2/512, ttl=64 (request in 385)
441	8.663483870	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request 1d=0x0003, seq=3/768, ttl=64 (reply in 442)
442	8.665896773	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply 1d=0x0003, seq=3/768, ttl=64 (request in 441)
545	9.665897372	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request 1d=0x0003, seq=4/1024, ttl=64 (reply in 546)
546	9.668876120	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply 1d=0x0003, seq=4/1024, ttl=64 (request in 545)
593	10.666274025	172.26.41.9	193.136.9.240	ICMP	1516	Echo (ping) request 1d=0x0003, seq=5/1280, ttl=64 (reply in 594)
594	10.668521593	193.136.9.240	172.26.41.9	ICMP	1516	Echo (ping) reply 1d=0x0003, seq=5/1280, ttl=64 (request in 593)

Fig.12. Mensagem com tamanho de 1480, não precisando de ser fragmentada, através da execução do comando *ping -s 1472 marco.uminho.pt*

1500	33.573182416	172.26.41.9	193.136.9.240	ICMP	1516 Echo (ping) request id=0x0004, seq=1/256, ttl=64 (reply in 1514)
1510	33.573205181	172.26.41.9	193.136.9.240	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=7400)
1513	33.573680703	193.136.9.240	172.26.41.9	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=262d)
1514	33.5736571081	193.136.9.240	172.26.41.9	ICMP	1516 Echo (ping) reply id=0x0004, seq=1/256, ttl=61 (request in 1500)
1533	34.574676647	172.26.41.9	193.136.9.240	ICMP	1516 Echo (ping) request id=0x0004, seq=2/512, ttl=64 (reply in 1532)
1534	34.574707652	172.26.41.9	193.136.9.240	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=7400)
1551	34.592354135	193.136.9.240	172.26.41.9	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=2840)
1552	34.592355042	193.136.9.240	172.26.41.9	ICMP	1516 Echo (ping) reply id=0x0004, seq=2/512, ttl=61 (request in 1533)
1704	35.575962310	172.26.41.9	193.136.9.240	ICMP	1516 Echo (ping) request id=0x0004, seq=3/768, ttl=64 (reply in 1704)
1702	35.576901276	172.26.41.9	193.136.9.240	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=74f0)
1703	35.576146851	193.136.9.240	172.26.41.9	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=291f)
1704	35.578474151	193.136.9.240	172.26.41.9	ICMP	1516 Echo (ping) reply id=0x0004, seq=3/768, ttl=61 (request in 1701)
1729	36.577518006	172.26.41.9	193.136.9.240	ICMP	1516 Echo (ping) request id=0x0004, seq=4/1024, ttl=64 (reply in 1705)
1730	36.577553271	172.26.41.9	193.136.9.240	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=75d4)
1763	36.634285569	193.136.9.240	172.26.41.9	IPv4	37 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=29d2)

Fig.13. Mensagem com tamanho de 1481, precisando de ser fragmentada, através da execução do comando *ping -s 1473 marco.uminho.pt*

Com isto, conseguimos concluir então que o tamanho máximo de uma mensagem, de forma a não ser necessário de fragmentar, é igual a 1472. Isto deve-se ao facto de o tamanho do cabeçalho ser igual a 20 e o do ID ser igual a 8 e, com isto, sabemos que mensagens com SIZE maior que 1472 serão fragmentadas pois $1473 + 20 + 8 = 1501$, ultrapassando o valor do MTU e sendo necessário fragmentar.

Parte 2 - Protocolo IPv4 :: Endereçamento e Encaminhamento IP

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.

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

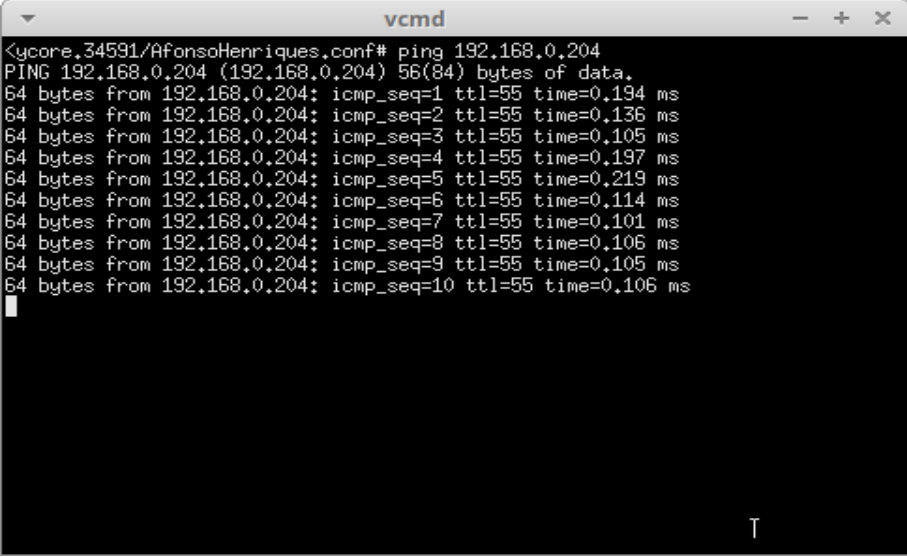
Finanças:

```

vcmd
rtt min/avg/max/mdev = 0.094/0.124/0.184/0.029 ms
<core.34591/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=0.050 ms
64 bytes from 192.168.0.250: icmp_seq=2 ttl=61 time=0.070 ms
64 bytes from 192.168.0.250: icmp_seq=3 ttl=61 time=0.077 ms
64 bytes from 192.168.0.250: icmp_seq=4 ttl=61 time=0.064 ms
64 bytes from 192.168.0.250: icmp_seq=5 ttl=61 time=0.081 ms
64 bytes from 192.168.0.250: icmp_seq=6 ttl=61 time=0.066 ms
64 bytes from 192.168.0.250: icmp_seq=7 ttl=61 time=0.064 ms
64 bytes from 192.168.0.250: icmp_seq=8 ttl=61 time=0.065 ms
64 bytes from 192.168.0.250: icmp_seq=9 ttl=61 time=0.086 ms
64 bytes from 192.168.0.250: icmp_seq=10 ttl=61 time=0.075 ms
64 bytes from 192.168.0.250: icmp_seq=11 ttl=61 time=0.063 ms
64 bytes from 192.168.0.250: icmp_seq=12 ttl=61 time=0.062 ms
64 bytes from 192.168.0.250: icmp_seq=13 ttl=61 time=0.064 ms
64 bytes from 192.168.0.250: icmp_seq=14 ttl=61 time=0.060 ms
64 bytes from 192.168.0.250: icmp_seq=15 ttl=61 time=0.064 ms
64 bytes from 192.168.0.250: icmp_seq=16 ttl=61 time=0.061 ms
64 bytes from 192.168.0.250: icmp_seq=17 ttl=61 time=0.085 ms
64 bytes from 192.168.0.250: icmp_seq=18 ttl=61 time=0.064 ms
64 bytes from 192.168.0.250: icmp_seq=19 ttl=61 time=0.064 ms
64 bytes from 192.168.0.250: icmp_seq=20 ttl=61 time=0.067 ms
64 bytes from 192.168.0.250: icmp_seq=21 ttl=61 time=0.065 ms

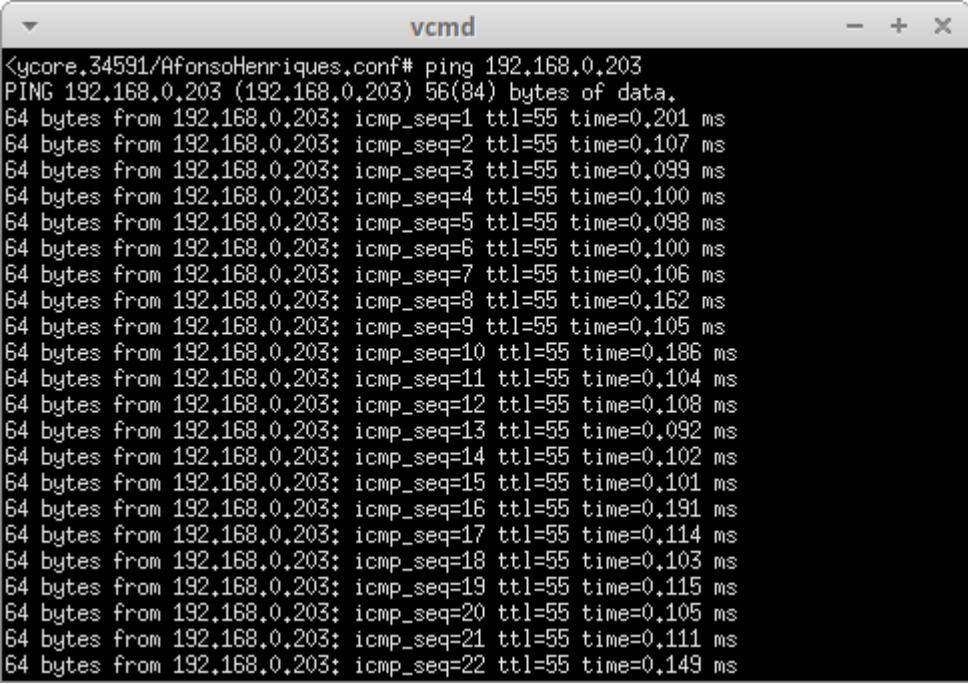
```

HBO:



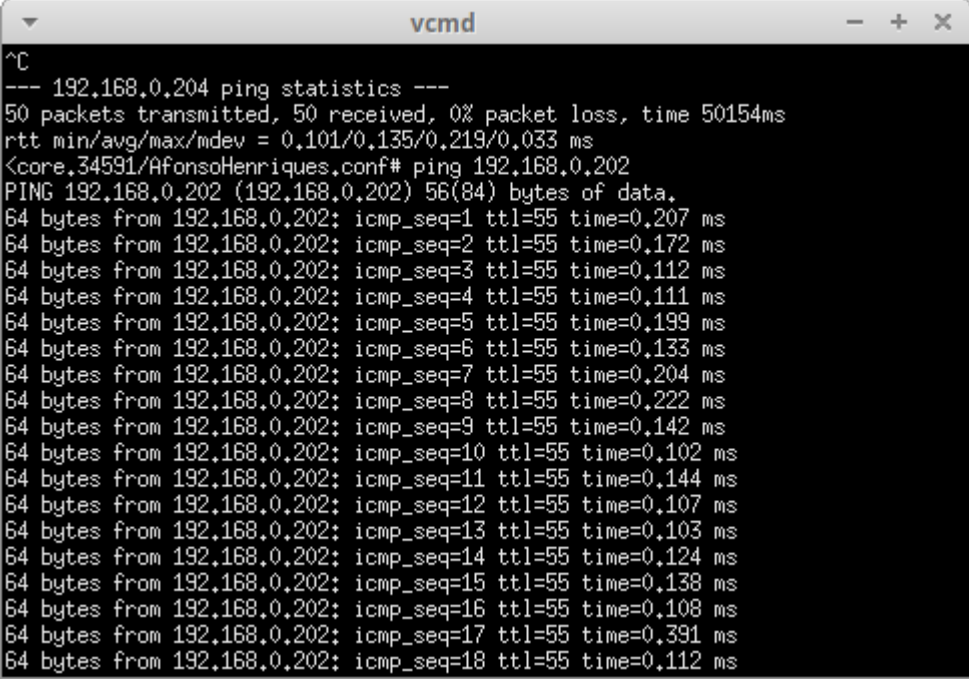
```
<ycore.34591/AfonsoHenriques.conf# ping 192.168.0.204
PING 192.168.0.204 (192.168.0.204) 56(84) bytes of data.
64 bytes from 192.168.0.204: icmp_seq=1 ttl=55 time=0.194 ms
64 bytes from 192.168.0.204: icmp_seq=2 ttl=55 time=0.136 ms
64 bytes from 192.168.0.204: icmp_seq=3 ttl=55 time=0.105 ms
64 bytes from 192.168.0.204: icmp_seq=4 ttl=55 time=0.197 ms
64 bytes from 192.168.0.204: icmp_seq=5 ttl=55 time=0.219 ms
64 bytes from 192.168.0.204: icmp_seq=6 ttl=55 time=0.114 ms
64 bytes from 192.168.0.204: icmp_seq=7 ttl=55 time=0.101 ms
64 bytes from 192.168.0.204: icmp_seq=8 ttl=55 time=0.106 ms
64 bytes from 192.168.0.204: icmp_seq=9 ttl=55 time=0.105 ms
64 bytes from 192.168.0.204: icmp_seq=10 ttl=55 time=0.106 ms
```

Netflix:



```
<ycore.34591/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=0.201 ms
64 bytes from 192.168.0.203: icmp_seq=2 ttl=55 time=0.107 ms
64 bytes from 192.168.0.203: icmp_seq=3 ttl=55 time=0.099 ms
64 bytes from 192.168.0.203: icmp_seq=4 ttl=55 time=0.100 ms
64 bytes from 192.168.0.203: icmp_seq=5 ttl=55 time=0.098 ms
64 bytes from 192.168.0.203: icmp_seq=6 ttl=55 time=0.100 ms
64 bytes from 192.168.0.203: icmp_seq=7 ttl=55 time=0.106 ms
64 bytes from 192.168.0.203: icmp_seq=8 ttl=55 time=0.162 ms
64 bytes from 192.168.0.203: icmp_seq=9 ttl=55 time=0.105 ms
64 bytes from 192.168.0.203: icmp_seq=10 ttl=55 time=0.186 ms
64 bytes from 192.168.0.203: icmp_seq=11 ttl=55 time=0.104 ms
64 bytes from 192.168.0.203: icmp_seq=12 ttl=55 time=0.108 ms
64 bytes from 192.168.0.203: icmp_seq=13 ttl=55 time=0.092 ms
64 bytes from 192.168.0.203: icmp_seq=14 ttl=55 time=0.102 ms
64 bytes from 192.168.0.203: icmp_seq=15 ttl=55 time=0.101 ms
64 bytes from 192.168.0.203: icmp_seq=16 ttl=55 time=0.191 ms
64 bytes from 192.168.0.203: icmp_seq=17 ttl=55 time=0.114 ms
64 bytes from 192.168.0.203: icmp_seq=18 ttl=55 time=0.103 ms
64 bytes from 192.168.0.203: icmp_seq=19 ttl=55 time=0.115 ms
64 bytes from 192.168.0.203: icmp_seq=20 ttl=55 time=0.105 ms
64 bytes from 192.168.0.203: icmp_seq=21 ttl=55 time=0.111 ms
64 bytes from 192.168.0.203: icmp_seq=22 ttl=55 time=0.149 ms
```

Youtube:



```
vcmd
^C
--- 192.168.0.204 ping statistics ---
50 packets transmitted, 50 received, 0% packet loss, time 50154ms
rtt min/avg/max/mdev = 0.101/0.135/0.219/0.033 ms
<core.34591/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=0.207 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.172 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.112 ms
64 bytes from 192.168.0.202: icmp_seq=4 ttl=55 time=0.111 ms
64 bytes from 192.168.0.202: icmp_seq=5 ttl=55 time=0.199 ms
64 bytes from 192.168.0.202: icmp_seq=6 ttl=55 time=0.133 ms
64 bytes from 192.168.0.202: icmp_seq=7 ttl=55 time=0.204 ms
64 bytes from 192.168.0.202: icmp_seq=8 ttl=55 time=0.222 ms
64 bytes from 192.168.0.202: icmp_seq=9 ttl=55 time=0.142 ms
64 bytes from 192.168.0.202: icmp_seq=10 ttl=55 time=0.102 ms
64 bytes from 192.168.0.202: icmp_seq=11 ttl=55 time=0.144 ms
64 bytes from 192.168.0.202: icmp_seq=12 ttl=55 time=0.107 ms
64 bytes from 192.168.0.202: icmp_seq=13 ttl=55 time=0.103 ms
64 bytes from 192.168.0.202: icmp_seq=14 ttl=55 time=0.124 ms
64 bytes from 192.168.0.202: icmp_seq=15 ttl=55 time=0.138 ms
64 bytes from 192.168.0.202: icmp_seq=16 ttl=55 time=0.108 ms
64 bytes from 192.168.0.202: icmp_seq=17 ttl=55 time=0.391 ms
64 bytes from 192.168.0.202: icmp_seq=18 ttl=55 time=0.112 ms
```

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.



```
vcmd
root@AfonsoHenriques:/tmp/pycore.40065/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.40065/AfonsoHenriques.conf#
```

Não possui nenhum problema, as únicas entradas são do router que conecta os seus condados aos seus ISPs.

A primeira entrada é a default, utilizada para o encaminhamento para todos os destinos que não estão especificado. O Next Hop ou Gateway no caso de Afonso Henriques e Teresa são, respetivamente, o RACondado e o RAGaliza, uma vez que são os routers a que os hosts estão ligados e que estabelecem ligação com os restantes elementos.

A segunda entrada tem destino os dispositivos que se encontram na mesma rede que o dispositivo verificado e o valor do gateway é 0.0.0.0. Logo o “next hop” não está especificado, já que não é preciso passar para outros routers pois o dispositivo está conectado diretamente com o destino pretendido porque ambos estão na mesma rede.

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.

Depois de utilizar o Wireshark, pudemos ver que 10.0.0.29 manda uma “mensagem” para Afonso Henriques com a informação de que a network está unreachable.

Então, para corrigir, adicionamos rotas para a Teresa. Depois, voltamos a utilizar o Wireshark, mas outro erro ocorreu na 10.0.0.25. Nesta situação, corrigimos o path ao mudar um caminho para 10.0.0.13, onde ocorreu um `time limit exceeded`, causado por um loop no caminho, que foi corrigido ao trocar o path e, em vez de encaminhar a mensagem para trás(10.0.0.14), dirigi-la para 10.0.0.5. Em n3, removemos um caminho que poderia impedir a passagem correta da mensagem, pois enviá-la-ia para 10.0.0.18, que não é o caminho pretendido.

Depois, verificamos que conseguimos ver no ISP CondadOnline um request proveniente do Afonso Henriques, no wireshark.

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.

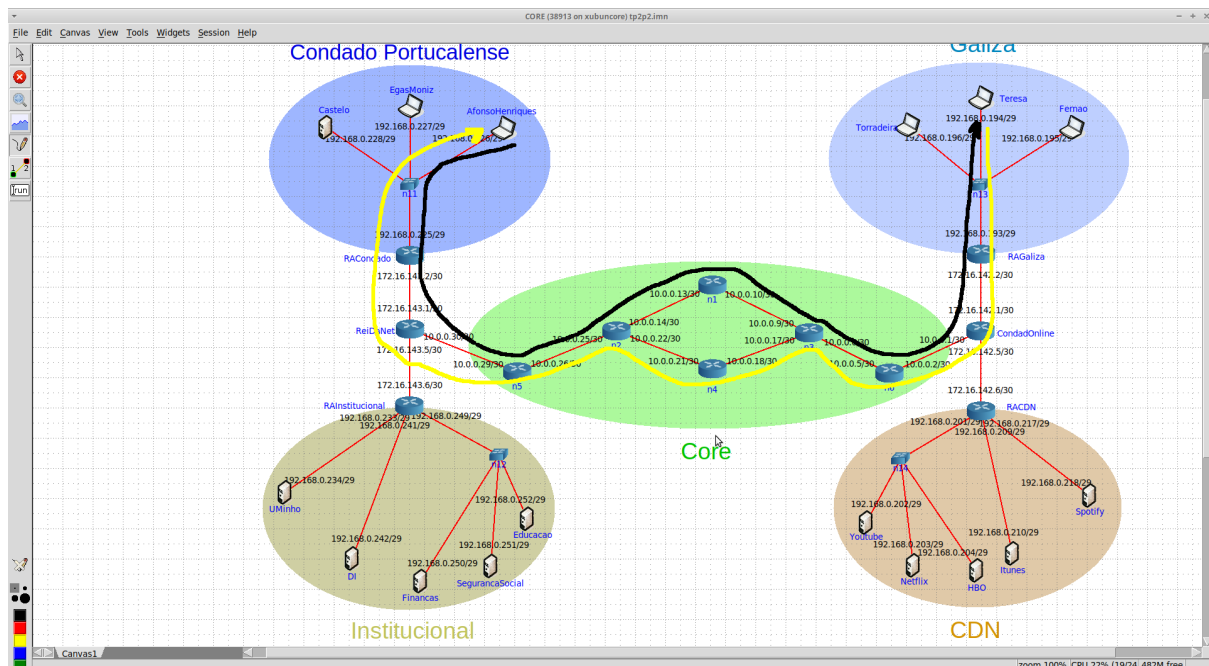
Teresa recebe os pacotes enviados por Afonso Henriques. Apesar do contrário não se verificar.

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.

Para o Afonso Henriques também receber os pacotes de volta, adicionamos um caminho em que, para enviar pacotes de Teresa para Afonso Henriques, o router de RAGaliza encaminha os pacotes para o ISP CondadOnline, algo que não fazia antes.

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.

As rotas de Afonso para Teresa não são as mesmas que a rota entre Teresa e Afonso.



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.

Não existe match nem quando são enviados pacotes para a RAGaliza nem quando são enviados pacotes para CDN. Esta entrada não é utilizada porque entra em ciclo.

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

Tanto aqueles utilizados pelos quatro polos como no core são privados, pois segundo a norma RFC1918, os mesmos estão contidos nos intervalos de endereços privados (entre 10.0.0.0 a 10.255.255.255, 172.16.0.0 a 172.31.255.255 e 192.168.0.0 a 192.168.255.255).

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

Como os switches localizados em cada um dos polos trabalham sobre a camada 2 (Data Link Layer), eles não possuem endereço IP atribuído.

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

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.

Uma rota default é importante pois é o caminho de pacotes quando o dispositivo não conhece a origem desses pacotes.

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.

IP-> 172.16.14.128/26.

32-26 = 6.

Mínimo de 3 redes -> 2 bits para a sub-rede.

Host -> 4 bits.

Máscara de rede original /26.

Máscara de rede para subnetting /28 -> 2 bits para a sub-rede (já definidos).

Sub-rede	Host	IPs
00	0001 - 1110	172.16.14.10000001(129) - 172.16.14.10001110(142)
01	0001 - 1110	172.16.14.10010001(145) - 172.16.14.10011110(158)
10	0001 - 1110	172.16.14.10100001(161) - 172.16.14.10101110(174)

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?

O Castelo não se conseguiu conectar, enquanto que todos os outros hosts conseguiram, visto que foi removida a rota default do Castelo.

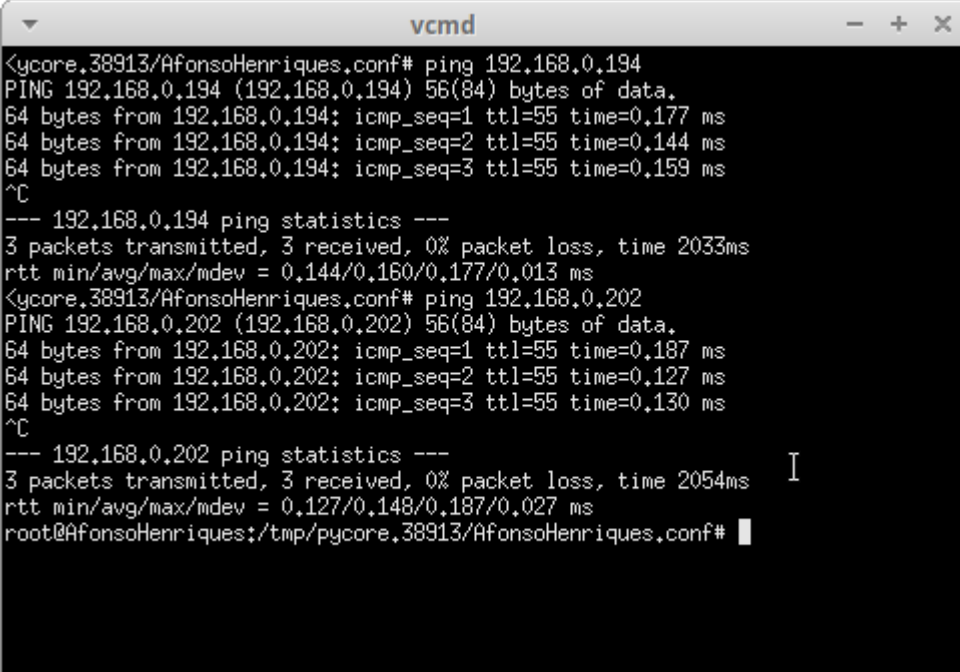
3. Ao planejar um novo ataque, D. Afonso Henriques constata que o seu exército não só perde bastante tempo a decidir que direção tomar a cada salto como, por vezes, inclusivamente se perde.

- 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.**

Primeiro, foram removidas as rotas do n6 para os hosts da galiza e CDN.

```
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.192/29
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.200/29
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.208/29
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.216/29
```

Depois, foi adicionado uma rota para 192.168.0.192/27 pois ele apanha todas as redes que foram eliminadas, visto que elas possuíam 192.168.0.110(3 primeiros bits) em comum, logo vai ter uma máscara de 27 bits.

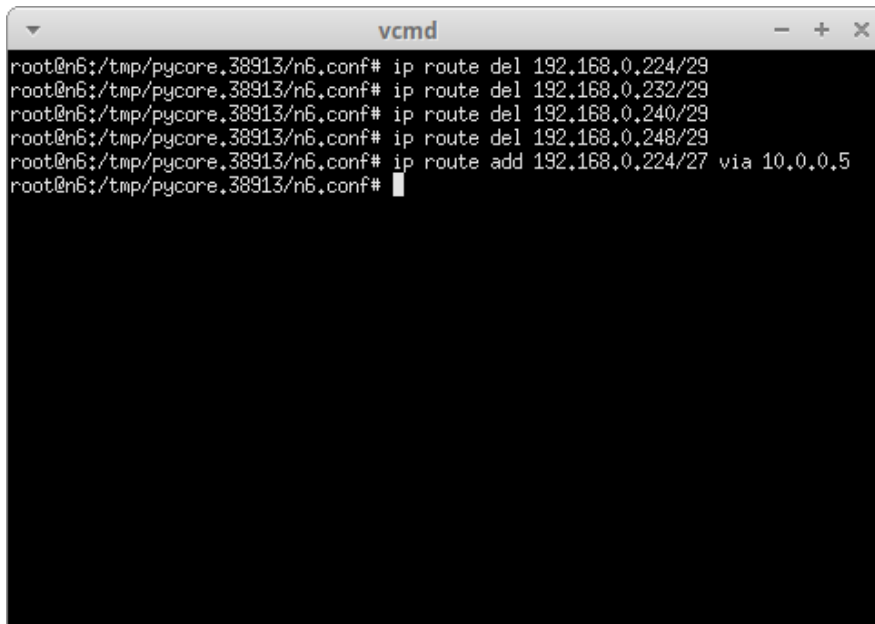


```
vcmd
<ycore.38913/AfonsoHenriques.conf# ping 192.168.0.194
PING 192.168.0.194 (192.168.0.194) 56(84) bytes of data.
64 bytes from 192.168.0.194: icmp_seq=1 ttl=55 time=0.177 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=55 time=0.144 ms
64 bytes from 192.168.0.194: icmp_seq=3 ttl=55 time=0.159 ms
^C
--- 192.168.0.194 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2033ms
rtt min/avg/max/mdev = 0.144/0.160/0.177/0.013 ms
<ycore.38913/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=0.187 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.127 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.130 ms
^C
--- 192.168.0.202 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.127/0.148/0.187/0.027 ms
root@AfonsoHenriques:/tmp/pycore.38913/AfonsoHenriques.conf#
```

Como podemos ver, a conectividade manteve-se.

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

Tal como na alínea anterior apagamos 4 entradas da tabela de encaminhamento e acrescentamos uma que as substitua.



```
vcmd
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.224/29
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.232/29
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.240/29
root@n6:/tmp/pycore.38913/n6.conf# ip route del 192.168.0.248/29
root@n6:/tmp/pycore.38913/n6.conf# ip route add 192.168.0.224/27 via 10.0.0.5
root@n6:/tmp/pycore.38913/n6.conf#
```

c) Comente os aspetos positivos e negativos do uso do Supernetting.

Supernetting é um processo inverso de sub-redes, no qual várias redes são mescladas em uma única rede, devido a isso tal processo tem diversas desvantagens e vantagens.

->Vantagens

- O tamanho da tabela de memória do roteador é minimizado resumindo várias entradas de informações de roteamento em uma única entrada, tornando as operações de rede mais rápidas e eficientes.
- Redução do tráfego da rede.
- Facilidade na administração da rede: o gerenciamento de várias sub-redes pode ser complicado e demorado, enquanto que a agregação em blocos maiores pode simplificar a administração da rede.
- Melhoria da segurança, pois quando sub-redes são agregadas, tornam-se mais difíceis de serem identificadas, o que leva a serem mais dificilmente alvos de ataques.

->Desvantagens

- A combinação de blocos deve ser feita no poder 2; alternativamente, se os três blocos são necessários, então deve ser atribuído quatro blocos.
- As redes têm de existir todas na mesma classe.
- Maior complexidade na configuração e manutenção.
- Risco de falha em um único ponto, pois se uma rede maior que foi criada por meio do Supernetting falhar, todas as redes menores que foram agrupadas nela também falharão;