



Universidade do Minho
Escola de Engenharia

Redes de Computadores

Licenciatura em Engenharia Informática

Ano Letivo de 2023/2024

TP2

Diogo Gabriel Lopes Miranda (a100839)

João Ricardo Ribeiro Rodrigues (a100598)

Délio Miguel Lopes Alves(a94557)

RC

Índice

| | |
|-------------------------------------|----|
| Parte 1- Questões e Respostas | 4 |
| Exercício 1 | 4 |
| Questão1- | 4 |
| 1.1 Alínea a -..... | 5 |
| 1.1 Alínea b..... | 7 |
| 1.1 Alínea c | 7 |
| 1.1 Alínea d..... | 8 |
| Exercício 2 | 9 |
| Questão2-..... | 9 |
| 1.2 Alínea a..... | 9 |
| 1.2 Alínea b..... | 10 |
| 1.2 Alínea c..... | 10 |
| 1.2 Alínea d..... | 11 |
| 1.2 Alínea e..... | 12 |
| 1.2 Alínea f | 13 |
| 1.2 Alínea g..... | 13 |
| 1.2 Alínea h..... | 14 |
| Exercício 3..... | 14 |
| 1.3. Alínea a | 14 |
| 1.3. Alínea b | 15 |
| 1.3. Alínea c | 15 |
| 1.3. Alínea d | 16 |
| 1.3. Alínea e | 16 |
| 1.3. Alínea f..... | 17 |
| 1.3. Alínea g | 18 |
| 1.3. Alínea h | 18 |
| 1.3. Alínea i | 18 |
| 1.3. Alínea j | 19 |
| Parte 2- Questões e Respostas..... | 20 |
| Exercício 1 | 21 |
| 2.1. Alínea a | 21 |
| 2.1. Alínea b | 22 |
| 2.1. Alínea c | 23 |
| Exercício 2..... | 24 |
| 2.2. Alínea a | 25 |

| | |
|--------------------------------|----|
| 2.2. Alínea b | 25 |
| 2.2. Alínea c | 26 |
| 2.2. Alínea d | 30 |
| 2.2.d.i | 30 |
| 2.2.d.ii | 32 |
| 2.2. Alínea e | 34 |
| 2.2. Alínea f | 34 |
| 2.2. Alínea g | 35 |
| Exercício 3 | 35 |
| 2.3. Alínea A e Alínea B | 35 |
| 2.3. Alínea c | 37 |

Parte 1- Questões e Respostas

Exercício 1

Questão1- Prepare uma topologia CORE para verificar o comportamento do traceroute. Na topologia deve existir: um host (pc) cliente designado Jasmine, cujo router de acesso é RA1; o router RA1 está simultaneamente ligado a dois routers no core da rede RCx e RCy (cada grupo de trabalho deve personalizar a rede de core fazendo x e y corresponder ao seu identificador de grupo PLxy); estes estão conectados a um router de acesso RA2, que por sua vez, se liga a um host (servidor) designado Aladdin. Ajuste o nome dos equipamentos atribuídos por defeito para o enunciado. Apenas nas ligações (links) da rede de core, estabeleça um tempo de propagação de 20 ms. Após ativar a topologia, note que pode não existir conectividade IP imediata entre Jasmine e Aladdin pois é necessário que o anúncio de rotas entre routers se efetue e estabilize.

O traceroute é utilizado para contruir o caminho que um determinado pacote faz desde a sua origem até ao seu destinatário, para tal, ele envia vários pacotes com o incremento do TTL(Time-To-live), isto é, a cada o destinatário envia uma mensagem com TTL igual a 1 e quando essa mensagem chegar ao primeiro router esse valor decresce uma unidade e quando chegar a 0 é enviada uma mensagem a informar que excedeu o TTL, incrementando o TTL em 1 unidade e repetindo o processo. Este processo, acontece até o caminho desde a origem até a fim estiver totalmente descoberto.

Como indicado no enunciado, preparamos uma topologia CORE, com um host (PC) com o nome Jasmine. Ligámo-lo ao router de acesso RA1. Este, por sua vez, está ligado aos routers RC6 e RC0. Estes dois routers encontram-se ligados ao router de acesso RA2, que por fim, se liga a um host (servidor), com o nome Aladdin.

Por último, nas ligações da rede foi estabelecido um tempo de propagação de 20ms.

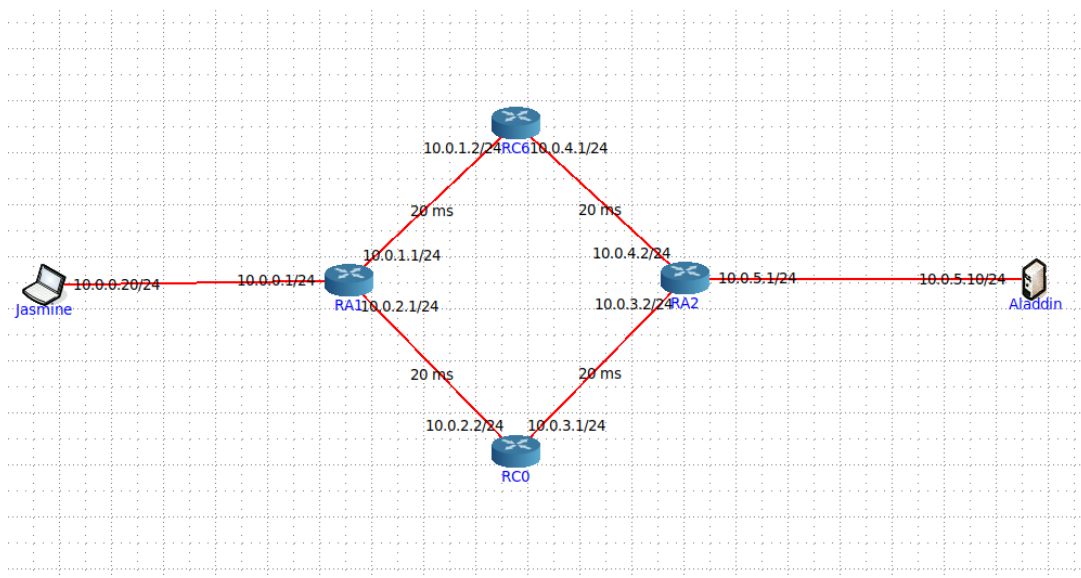


Figura 1 - Topologia utilizada

1.1 Alínea a - Active o wireshark no host Jasmine. Numa shell de Jasmine execute o comando *traceroute -I* para o endereço IP do Aladdin. Registe e analise o tráfego ICMP enviado pelo sistema Jasmine e o tráfego ICMP recebido como resposta. Explique os resultados obtidos tendo em conta o princípio de funcionamento do traceroute

De forma a analisar o tráfego recebido e enviado resultante da execução do comando *traceroute -I*, utilizamos o Wireshark para capturar o tráfego resultante. As informações obtidas estão apresentadas na *Figura 2*.

O host *Jasmine* tem IP: 10.0.0.20 (origem) e o host *Aladdin* tem IP: 10.0.5.10 (destinatário).

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|--------------|------------------------|-------------------|----------|--------|--|
| 1 | 0.000000000 | 10.0.0.1 | 224.0.0.5 | OSPF | 78 | Hello Packet |
| 2 | 1.460866154 | 00:00:00_aa:00:0a | Broadcast | ARP | 42 | Who has 10.0.0.1? Tell 10.0.0.20 |
| 3 | 1.460893480 | 00:00:00_aa:00:0b | 00:00:00_aa:00:0a | ARP | 42 | 10.0.0.1 is at 00:00:00_aa:00:0b |
| 4 | 1.460895222 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=1/256, ttl=1 (no response... |
| 5 | 1.460900674 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 6 | 1.460911910 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=2/512, ttl=1 (no response... |
| 7 | 1.460914274 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 8 | 1.460916337 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=3/768, ttl=1 (no response... |
| 9 | 1.460918210 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 10 | 1.460920214 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=4/1024, ttl=2 (no respons... |
| 11 | 1.460928678 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=5/1280, ttl=2 (no respons... |
| 12 | 1.460931092 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=6/1536, ttl=2 (no respons... |
| 13 | 1.460933485 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=7/1792, ttl=3 (no respons... |
| 14 | 1.460935489 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no respons... |
| 15 | 1.460937443 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no respons... |
| 16 | 1.460939666 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in ... |
| 17 | 1.460941649 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in ... |
| 18 | 1.460943613 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (reply in ... |
| 19 | 1.460945696 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (reply in ... |
| 20 | 1.460947629 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (reply in ... |
| 21 | 1.460949542 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (reply in ... |
| 22 | 1.460951596 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (reply in ... |
| 23 | 1.461300233 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=17/4352, ttl=6 (reply in ... |
| 24 | 1.461307235 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=18/4608, ttl=6 (reply in ... |
| 25 | 1.461310040 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=19/4864, ttl=7 (reply in ... |
| 26 | 1.506262137 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 27 | 1.506266213 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 28 | 1.506267035 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 29 | 1.506639251 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=20/5120, ttl=7 (reply in ... |
| 30 | 1.506652412 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=21/5376, ttl=7 (reply in ... |
| 31 | 1.506657381 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x001b, seq=22/5632, ttl=8 (reply in ... |
| 32 | 1.5509989601 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 33 | 1.550994169 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 34 | 1.550995040 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 35 | 1.550995732 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=10/2560, ttl=61 (request ... |
| 36 | 1.550996373 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=11/2816, ttl=61 (request ... |
| 37 | 1.550997034 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=12/3072, ttl=61 (request ... |
| 38 | 1.550997675 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=13/3328, ttl=61 (request ... |
| 39 | 1.550998316 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=14/3584, ttl=61 (request ... |
| 40 | 1.550998967 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=15/3840, ttl=61 (request ... |
| 41 | 1.550999598 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=16/4096, ttl=61 (request ... |
| 42 | 1.551000239 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=17/4352, ttl=61 (request ... |
| 43 | 1.551000890 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=18/4608, ttl=61 (request ... |
| 44 | 1.551001461 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=19/4864, ttl=61 (request ... |
| 45 | 1.594562085 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=20/5120, ttl=61 (request ... |
| 46 | 1.594567073 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=21/5376, ttl=61 (request ... |
| 47 | 1.594567944 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x001b, seq=22/5632, ttl=61 (request ... |
| 48 | 2.003050690 | 10.0.0.1 | 224.0.0.5 | OSPF | 78 | Hello Packet |
| 49 | 2.466015737 | fe80::200:ff:feaa:b | ff02::5 | OSPF | 90 | Hello Packet |
| 50 | 2.631195367 | fe80::3435:88ff:fe9... | ff02::fb | MDNS | 107 | Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR... |
| 51 | 3.212437812 | fe80::44b5:5aff:fe8... | ff02::fb | MDNS | 107 | Standard query 0x0000 PTR _ipps._tcp.local, "QM" question PTR... |
| 52 | 4.003497190 | 10.0.0.1 | 224.0.0.5 | OSPF | 78 | Hello Packet |

Figura 2 - Output wireshark do traceroute de Jasmin para Aladin

Após uma análise inicial do output do Wireshark podemos observar que, por predefinição, são sempre enviados três pacotes em cada tentativa de atingir o seu destino.

Cada pacote possui um campo denominado TTL, Time To Live, que indica o número de *routers* que um pacote pode atravessar antes de ser descartado. Caso o número não seja suficiente para chegar ao seu destino, é enviada uma mensagem de erro (com origem no *router* onde chegou) indicando que o TTL foi excedido. As linhas com fundo preto no output representam essas situações.

Cada pacote tem um valor de TTL específico (mas igual nos três pacotes que são enviados do mesmo local). No primeiro pacote o valor é TTL=1. Este não é suficiente para atingir o destino, e como tal recebe uma mensagem de erro (TTL excedido) tal mostra a figura abaixo:

| | | | | | | |
|---|-------------|-----------|-----------|------|---------------------------|--|
| 3 | 3.176246985 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=1/256, ttl=1 (no response found!) |
| 4 | 3.176273372 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 5 | 3.176279662 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=2/512, ttl=1 (no response found!) |
| 6 | 3.176282547 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 7 | 3.176284952 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=3/768, ttl=1 (no response found!) |
| 8 | 3.176287116 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |

Figura 3 – Pacotes com TTL 1 e mensagem de erro

Como não conseguiu atingir o seu destino com TTL=1, o valor deste será incrementado em uma unidade até que seja recebida uma resposta (*reply*). Na figura abaixo podemos observar o envio de pacotes com TTL igual a 2, 3, 4, 5, 6, 7 e 8 de forma seguida. Isto ocorre, pois, após o envio dos primeiros pacotes o *host* conclui que enviar todos os pacotes seguidos e parar quando receber uma *reply* de sucesso é mais eficiente do que enviar apenas após receber a mensagem de erro.

| | | | | | | |
|----|-------------|-----------|-----------|------|---------------------------|---|
| 9 | 3.176289499 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=4/1024, ttl=2 (no response found!) |
| 10 | 3.176298325 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=5/1280, ttl=2 (no response found!) |
| 11 | 3.176301089 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=6/1536, ttl=2 (no response found!) |
| 12 | 3.176303814 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=7/1792, ttl=3 (no response found!) |
| 13 | 3.176306129 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=8/2048, ttl=3 (no response found!) |
| 14 | 3.176308393 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=9/2304, ttl=3 (no response found!) |
| 15 | 3.176310907 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=10/2560, ttl=4 (reply in 34) |
| 16 | 3.176313211 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=11/2816, ttl=4 (reply in 35) |
| 17 | 3.176315465 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=12/3072, ttl=4 (reply in 36) |
| 18 | 3.176317869 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=13/3328, ttl=5 (reply in 37) |
| 19 | 3.176320063 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=14/3584, ttl=5 (reply in 38) |
| 20 | 3.176322237 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=15/3840, ttl=5 (reply in 39) |
| 21 | 3.176324641 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=16/4096, ttl=6 (reply in 40) |
| 22 | 3.176675744 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=17/4352, ttl=6 (reply in 41) |
| 23 | 3.176683708 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=18/4608, ttl=6 (reply in 42) |
| 24 | 3.176687103 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=19/4864, ttl=7 (reply in 43) |
| 25 | 3.220249887 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 26 | 3.220254384 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 27 | 3.220255146 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 28 | 3.220724385 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=20/5120, ttl=7 (reply in 44) |
| 29 | 3.220740122 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=21/5376, ttl=7 (reply in 45) |
| 30 | 3.220746093 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 Echo (ping) request | id=0x001c, seq=22/5632, ttl=8 (reply in 46) |
| 31 | 3.262040997 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 32 | 3.262045595 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 33 | 3.262046447 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 Time-to-live exceeded | (Time to live exceeded in transit) |
| 34 | 3.262047128 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 Echo (ping) reply | id=0x001c, seq=10/2560, ttl=61 (request in 15) |
| 35 | 3.262047809 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 Echo (ping) reply | id=0x001c, seq=11/2816, ttl=61 (request in 16) |
| 36 | 3.262048531 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 Echo (ping) reply | id=0x001c, seq=12/3072, ttl=61 (request in 17) |

Figura 4 - Restantes TTL's e reply a indicar sucesso

Na linha 34 verificamos que é recebido um reply, indicando que o request ao qual corresponde se encontra na linha 15, o que significa que o primeiro pacote a atingir o destino tinha um TTL=4 (verificando o IP de origem do *reply* podemos também confirmar que é proveniente do IP que pretendíamos atingir). Na mensagem de *reply* observamos que o TTL=61. Este valor é utilizado como uma segurança para garantir que a mensagem atinge o seu destino, sendo dado um TTL inflacionado superior. O valor pode variar consoante o fabricante do dispositivo a partir do qual foi enviado.

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

Como se pode verificar pela *Figura 4*, quando o TTL tem os valores 1, 2 e 3 não é possível alcançar o servidor, resultando em um erro informando que o TTL foi excedido. Apenas quando o TTL atinge o valor 4 é que obtemos um *reply*. Analisando também a figura abaixo, o caminho apresentado para atingir o destino possui 4 etapas. Com base nestas informações, podemos concluir que o valor inicial mínimo do TTL é 4

```
root@Jasmine:/tmp/pycore.39763/Jasmine.conf# traceroute -l 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)  0.060 ms  0.006 ms  0.005 ms
 2  10.0.1.2 (10.0.1.2)  40.886 ms  40.874 ms  40.871 ms
 3  10.0.3.2 (10.0.3.2)  81.360 ms  81.358 ms  81.354 ms
 4  10.0.5.10 (10.0.5.10)  81.350 ms  81.348 ms  81.345 ms
root@Jasmine:/tmp/pycore.39763/Jasmine.conf#
```

Figura 5 - Resultado do traceroute

1.1 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 RTT (Round-Trip Time) é o tempo que um pacote demora a ir de um ponto a outro da rede e retornar ao ponto de origem.

De forma a calcular o valor médio do RTT no acesso ao servidor *Aladin*, com origem no *host Jasmine*, utilizamos os valores fornecidos pela execução do comando *traceroute -q 5 -l 10.0.5.10* (semelhante ao comando utilizado anteriormente, mas neste caso envia 5 pacotes de prova para cada endereço). Estes valores estão apresentados na figura abaixo:

```
root@Jasmine:/tmp/pycore.39593/Jasmine.conf# traceroute -q 5 -l 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)  0.044 ms  0.004 ms  0.003 ms  0.003 ms  0.003 ms
 2  10.0.1.2 (10.0.1.2)  42.938 ms  42.931 ms  42.928 ms  42.926 ms  42.925 ms
 3  10.0.3.2 (10.0.3.2)  84.810 ms  84.808 ms  84.807 ms  84.805 ms  84.803 ms
 4  10.0.5.10 (10.0.5.10)  84.802 ms  85.672 ms  85.657 ms  85.655 ms  85.653 ms
root@Jasmine:/tmp/pycore.39593/Jasmine.conf#
```

Figura 6-Resultado do traceroute com 5 pacotes (-q 5)

Na linha 4 encontram-se os valores correspondentes ao RTT, de cada pacote, para o servidor Aladdin (IP: 10.0.5.10). A média desses 5 valores é igual a 85,4878 milissegundos.

$$RTT \text{ médio} = 85,4878 \text{ ms}$$

1.1 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 valor do RTT representa o tempo demorado para o pacote chegar ao destino e voltar ao seu local de origem. Tendo em conta essa lógica, podemos ser levados a pensar que dividindo o valor por dois obtemos o valor de ida e o de volta, *One-Way Delay*. No entanto, isso só aconteceria numa rede perfeita, onde nada existe que possa influenciar esses valores, o que não é caso das redes reais.

Existem dois fatores principais que influenciam o tempo de ida e volta, a congestão da rede e o caminho utilizado em ambos os percursos.

A congestão da rede provoca atrasos no percurso do pacote. Um maior tráfego ou uma fila de espera para aceder a um router num determinado momento pode fazer com que o tempo que o pacote demore na ida seja diferente de na volta, e vice-versa.

A escolha do caminho a utilizar é feita salto a salto, analisando as opções de próximos saltos existentes e escolhendo a melhor naquele momento. O facto de a escolha ser no momento faz com que nem sempre o caminho escolhido na ida seja igual ao da volta, pois devido a diversos fatores, como, por exemplo, a congestão, pode ser benéfico escolher uma rota diferente. Diferentes caminhos poderão ter diferentes tempos.

Tendo em conta estas duas situações, podemos concluir que não é possível calcular o *One-Way Delay* dividindo o RTT por dois, numa rede real.

Exercício 2

Questão2- 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 (expandir o tab correspondente na janela de detalhe do wireshark).

1.2 Alínea a.- Qual é o endereço IP da interface ativa do seu computador?

Após a execução do *traceroute* e da captura do tráfego resultante, identificamos a primeira mensagem ICMP e acedemos às suas informações, apresentadas na Figura 7.

```
▼ Internet Protocol Version 4, Src: 10.0.2.15, 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: 0x1cee (7406)
  ▶ Flags: 0x0000
    Fragment offset: 0
  ▶ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0xc54c [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 193.136.9.240
```

Figura 7 – Source da mensagem capturada

Analisando seu conteúdo podemos verificar que existe um campo *Source*, que indica o IP de origem da mensagem. Como é a primeira mensagem ICMP capturada, sabemos que a sua origem é o nosso computador. Assim sendo, podemos concluir que o IP da interface ativa do nosso computador é *10.0.2.15*.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-----------|---------------|----------|--------|--|
| 9 | 1.985662... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 | Echo (ping) request id=0x0001, seq=1/256, ttl=1 (no response found!) |
| 10 | 1.985705... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 | Echo (ping) request id=0x0001, seq=2/512, ttl=1 (no response found!) |
| 11 | 1.985748... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 | Echo (ping) request id=0x0001, seq=3/768, ttl=1 (no response found!) |

Figura 8 - Excerto da captura Wireshark (visualizar Source)

Seguindo o mesmo raciocínio, também poderíamos chegar à mesma resposta visualizando o campo *Source* apresentado na tabela de capturas do Wireshark (contém a mesma informação), na mensagem em questão, como apresentado na Figura 8.

1.2 Alínea b - Qual é o valor do campo *protocol*? O que permite identificar?

O valor do campo *Protocol* na primeira mensagem capturada é *Protocol: ICMP*, como podemos verificar na *Figura 9*. Este indica-nos o protocolo utilizado para transmitir a mensagem.

```
▼ Internet Protocol Version 4, Src: 10.0.2.15, 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: 0x1cee (7406)
  ▶ Flags: 0x0000
    Fragment offset: 0
  ▶ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0xc54c [validation disabled]
    [Header checksum status: Unverified]
  Source: 10.0.2.15
  Destination: 193.136.9.240
```

Figura 9 - Protocol da mensagem capturada

O protocolo ICMP é um protocolo da camada de rede, pertencente ao conjunto de protocolos TCP/IP, utilizado para comunicações de controlo e diagnóstico dentro de uma rede IP. Este é utilizado, por exemplo, para os comandos *ping* e *traceroute*, e para enviar mensagem de erro. No Exercício 1 vimos alguns exemplos da sua utilização no *traceroute* e para as mensagens de erro.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-----------|---------------|----------|--------|--|
| 9 | 1.985662... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 | Echo (ping) request id=0x0001, seq=1/256, ttl=1 (no response found!) |
| 10 | 1.985705... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 | Echo (ping) request id=0x0001, seq=2/512, ttl=1 (no response found!) |
| 11 | 1.985748... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 | Echo (ping) request id=0x0001, seq=3/768, ttl=1 (no response found!) |

Figura 10 - Excerto da captura Wireshark (visualizar Protocol)

Seguindo o mesmo raciocínio, também poderíamos chegar à mesma resposta visualizando o campo *Protocol* apresentado na tabela de capturas do Wireshark (contém a mesma informação), na mensagem em questão, como apresentado na *Figura 8*.

1.2 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?

Como podemos verificar na figura abaixo, o cabeçalho IPv4 possui 60 bytes, tal como indicado no campo *Total Length*.

```
▼ Internet Protocol Version 4, Src: 10.0.2.15, 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
```

Um datagrama possui um *header* e o *payload*. O *header* contém as informações necessárias para a entrega e interpretação dos dados contidos no pacote. É por isso algo informativo, não contendo a informação que era pretendida transmitir. O *payload* é a parte dos dados que contém a informação real que está a ser transmitida.

Assim sendo, para calcular o tamanho do *payload* devemos subtrair ao tamanho o datagrama o tamanho do *header*, ficando com a seguinte formula:

$$\text{Payload Length} = \text{Total Length} - \text{Header Length}$$

Tendo *Total Length*=60 bytes e *Header Length*=20 bytes, então o tamanho do *payload* é 20 bytes.

1.2 Alínea d_- O datagrama IP foi fragmentado? Justifique.

```
▼ Internet Protocol Version 4, Src: 10.0.2.15, 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: 0x5b8d (23437)
  ▼ 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: 0x86ad [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 193.136.9.240
```

Ao analisarmos os campos no cabeçalho do datagrama IP conseguimos observar que o valor no campo *Offset* é 0, o que pode indicar a ausência de fragmentação, ou então, tratar-se-ia do primeiro datagrama fragmentado, mas rapidamente anulamos esta hipótese, pois, a flag *More Fragments* não está ativada.

Assim, concluímos que o datagrama não foi fragmentado.

1.2 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.

| | | | | | | |
|----|-------------|-----------|---------------|------|--|--|
| 9 | 1.985662... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=1/256, ttl=1 (no response found!) |
| 10 | 1.985705... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=2/512, ttl=1 (no response found!) |
| 11 | 1.985748... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=3/768, ttl=1 (no response found!) |
| 12 | 1.985776... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=4/1024, ttl=2 (no response found!) |
| 13 | 1.985811... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=5/1280, ttl=2 (no response found!) |
| 14 | 1.986260... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=6/1536, ttl=2 (no response found!) |
| 15 | 1.986397... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=7/1792, ttl=3 (no response found!) |
| 19 | 1.986711... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=8/2048, ttl=3 (no response found!) |
| 20 | 1.987000... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=9/2304, ttl=3 (no response found!) |
| 21 | 1.987328... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=10/2560, ttl=4 (no response found!) |
| 22 | 1.987360... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=11/2816, ttl=4 (no response found!) |
| 23 | 1.987390... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=12/3072, ttl=4 (no response found!) |
| 24 | 1.987418... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=13/3328, ttl=5 (reply in 36) |
| 25 | 1.987444... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=14/3584, ttl=5 (reply in 37) |
| 26 | 1.987473... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=15/3840, ttl=5 (reply in 38) |
| 27 | 1.987499... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=16/4096, ttl=6 (reply in 39) |
| 28 | 1.988323... | 10.0.2.15 | 193.137.16.65 | DNS | 92 Standard query 0x09df PTR 2.2.0.10.in-addr.arpa OPT | |
| 43 | 1.993307... | 10.0.2.15 | 193.137.16.65 | DNS | 81 Standard query 0x09df PTR 2.2.0.10.in-addr.arpa | |
| 45 | 1.998650... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=17/4352, ttl=6 (reply in 48) |
| 46 | 1.998693... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=18/4608, ttl=6 (reply in 49) |
| 47 | 1.998724... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=19/4864, ttl=7 (reply in 50) |

Figura 11 - Pacotes ordenados pelo endereço IP fonte

Os pacotes gerados a partir do endereço IP da interface da nossa máquina são:

| | | | | | | |
|----|-------------|-----------|---------------|------|------------------------|---|
| 9 | 1.985662... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=1/256, ttl=1 (no response found!) |
| 10 | 1.985705... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=2/512, ttl=1 (no response found!) |
| 11 | 1.985748... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=3/768, ttl=1 (no response found!) |
| 12 | 1.985776... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=4/1024, ttl=2 (no response found!) |
| 13 | 1.985811... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=5/1280, ttl=2 (no response found!) |
| 14 | 1.986260... | 10.0.2.15 | 193.136.9.240 | ICMP | 74 Echo (ping) request | id=0x0001, seq=6/1536, ttl=2 (no response found!) |

Figura 12 - Pacotes com origem na nossa máquina

| | |
|--|--|
| <ul style="list-style-type: none"> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.240 <ul style="list-style-type: none"> 0100 ... = Version: 4 ... 0101 = Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x1cee (7406) Flags: 0x0000 Fragment offset: 0 Time to live: 1 Protocol: ICMP (1) Header checksum: 0xc54c [validation disabled] [Header checksum status: Unverified] Source: 10.0.2.15 Destination: 193.136.9.240 | <ul style="list-style-type: none"> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.240 <ul style="list-style-type: none"> 0100 ... = Version: 4 ... 0101 = Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x1cef (7407) Flags: 0x0000 Fragment offset: 0 Time to live: 1 Protocol: ICMP (1) Header checksum: 0xc54b [validation disabled] [Header checksum status: Unverified] Source: 10.0.2.15 Destination: 193.136.9.240 |
|--|--|

Figura 13 - TTL1 primeiro e segundo pacote

| |
|--|
| <ul style="list-style-type: none"> Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.240 <ul style="list-style-type: none"> 0100 ... = Version: 4 ... 0101 = Header Length: 20 bytes (5) Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 60 Identification: 0x1cf1 (7409) Flags: 0x0000 Fragment offset: 0 Time to live: 2 Protocol: ICMP (1) Header checksum: 0xc449 [validation disabled] [Header checksum status: Unverified] Source: 10.0.2.15 Destination: 193.136.9.240 |
|--|

Figura 14 - TTL2 primeiro pacote

Analisando as Figuras 13 e 14 verificamos que os campos que variam de pacote para pacote são o identificador, o TTL e checksum.

1.2 Alínea f - Observa algum padrão nos valores do campo de Identificação do datagrama IP e do TTL?

| | |
|--|--|
| <pre>Internet Protocol Version 4, Src: 10.0.2.15, 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: 0x1cee (7406) Flags: 0x0000 Fragment offset: 0 Time to live: 1 Protocol: ICMP (1) Header checksum: 0xc54c [validation disabled] [Header checksum status: Unverified] Source: 10.0.2.15 Destination: 193.136.9.240</pre> | <pre>Internet Protocol Version 4, Src: 10.0.2.15, 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: 0x1cef (7407) Flags: 0x0000 Fragment offset: 0 Time to live: 1 Protocol: ICMP (1) Header checksum: 0xc54b [validation disabled] [Header checksum status: Unverified] Source: 10.0.2.15 Destination: 193.136.9.240</pre> |
|--|--|

Figura 15 - TTL1 primeiro e segundo pacote

O campo de identificação aumenta um valor em cada pacote de forma sequencial para todos os pacotes recebidos.

O TTL aumenta um valor a cada três pacotes, porque o *traceroute* envia três pacotes com o mesmo TTL antes de aumentar e testar o próximo. Este valor é o valor padrão do comando no caso de não ser especificado outro usando *-q*.

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

1.2.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?

| | | | | | |
|----|-------------|----------------|-----------|------|---|
| 16 | 1.986497... | 10.0.2.2 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 17 | 1.986498... | 10.0.2.2 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 18 | 1.986498... | 10.0.2.2 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 29 | 1.988716... | 172.26.254.254 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 30 | 1.988841... | 172.26.254.254 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 31 | 1.989299... | 172.16.2.1 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 32 | 1.989299... | 172.26.254.254 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 33 | 1.990136... | 172.16.2.1 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 34 | 1.990137... | 172.16.2.1 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 35 | 1.990137... | 172.16.115.252 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 36 | 1.990473... | 193.136.9.240 | 10.0.2.15 | ICMP | 74 Echo (ping) reply id=0x0001, seq=13/3328, ttl=60 (req=13/3328) |
| 37 | 1.993128... | 193.136.9.240 | 10.0.2.15 | ICMP | 74 Echo (ping) reply id=0x0001, seq=14/3584, ttl=60 (req=14/3584) |
| 38 | 1.993128... | 193.136.9.240 | 10.0.2.15 | ICMP | 74 Echo (ping) reply id=0x0001, seq=15/3840, ttl=60 (req=15/3840) |
| 39 | 1.993128... | 193.136.9.240 | 10.0.2.15 | ICMP | 74 Echo (ping) reply id=0x0001, seq=16/4096, ttl=60 (req=16/4096) |
| 40 | 1.993128... | 172.16.115.252 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |
| 41 | 1.993128... | 172.16.115.252 | 10.0.2.15 | ICMP | 70 Time-to-live exceeded (Time to live exceeded in transit) |

Os valores de TTL recebidos são 255, 254 e 253.

O valor não permanece constante visto que, à medida que o TTL é excedido em um router, é mais uma ligação que o pacote vai ter de percorrer e, portanto o valor do TTL do *ICMP TTL Exceeded* decresce.

1.2.g.ii - Por que razão as mensagens de resposta ICMP TTL Exceeded são sempre enviadas na origem com um valor TTL relativamente alto?

São sempre enviadas com um valor alto para garantir que atingem o destino e evitar erros de exceder o TTL.

1.2 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?

A inclusão das informações do ICMP no cabeçalho do IPv4 é teoricamente viável, mas apresenta desvantagens e vantagens a serem consideradas. Por um lado, tal inclusão aumentaria o tamanho do cabeçalho do IPv4, que é fixo em 20 bytes, resultando em pacotes maiores. No entanto, a velocidade ganha compensa a complexidade acrescida. Além disso, os roteadores teriam de realizar mais processamento para analisar o cabeçalho do IPv4 e extrair as informações do ICMP, o que poderia causar atrasos na transmissão de pacotes. Outro problema seria que dessa forma seria sempre esperado um pacote ICMP. Caso não recebesse, então o pacote teria campos com lixos pois os campos ICMP seriam inúteis.

Por outro lado, há vantagens em incluir as informações do ICMP no cabeçalho do IPv4. Por exemplo, isso poderia reduzir o número de pacotes na rede, já que não seria necessário enviar pacotes separados para mensagens de controle e de erro. Além disso, a inclusão das informações do ICMP no cabeçalho do IPv4 poderia aumentar a confiabilidade, pois o pacote completo seria protegido pelo checksum do IPv4.

Exercício 3

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

Na execução do comando *traceroute* especificamos que o tamanho dos pacotes seria 3600 bytes. No entanto, apenas é possível transmitir 1500 bytes de cada vez na nossa topologia. Como tal, é necessário fragmentar o pacote de forma a respeitar esse tamanho limite.

1500 bytes de cada vez implica que o pacote seja dividido em 3 fragmentos, como é possível verificar na imagem abaixo.

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|---------------|---------------|----------|--------|---|
| 1 | 0.000000... | 10.0.2.15 | 193.137.16.65 | DNS | 86 | Standard query 0x8c5c AAAA marco.uminho.pt OPT |
| 2 | 0.004327... | 193.137.16.65 | 10.0.2.15 | DNS | 149 | Standard query response 0x8c5c AAAA marco.uminho.pt SOA dns.uminho.pt OPT |
| 3 | 0.005080... | 10.0.2.15 | 193.136.9.240 | ICMP | 15 | Echo (ping) request id=0x0004, seq=1/256, ttl=1 (no response found!) |
| 4 | 0.005163... | 10.0.2.15 | 193.136.9.240 | IPv4 | 15 | Fragmented IP protocol (proto=ICMP 1, off=1480, ID=dc00) |
| 5 | 0.005219... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dc00) |
| 6 | 0.005266... | 10.0.2.15 | 193.136.9.240 | ICMP | 15 | Echo (ping) request id=0x0004, seq=2/512, ttl=1 (no response found!) |
| 7 | 0.007755... | 10.0.2.15 | 193.136.9.240 | IPv4 | 15 | Fragmented IP protocol (proto=ICMP 1, off=1480, ID=dc01) |
| 8 | 0.007803... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dc01) |
| 9 | 0.007883... | 10.0.2.15 | 193.136.9.240 | ICMP | 15 | Echo (ping) request id=0x0004, seq=3/768, ttl=1 (no response found!) |

Figura 16 - Fragmentos da mensagem

1.3. 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?

O campo que nos indica que o datagrama foi fragmentado, no caso do primeiro fragmento, é o campo *more fragments*. Se *more fragments: set* então significa que mais fragmentos serão recebidos, o que nos indica que o datagrama foi fragmentado.

No caso do primeiro datagrama, é possível identificá-lo observando se *offset = 0*. Tendo estas informações em consideração, podemos verificar que o pacote presente na figura abaixo é o primeiro fragmento.

```
Internet Protocol Version 4, Src: 10.0.2.15, 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: 0xdcd0 (56528)
  Flags: 0x2000, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
  Fragment offset: 0
```

Figura 17 - Informações primeiro datagrama

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

Tal como podemos observar na figura abaixo, o valor do *offset* do segundo datagrama é 1480 bytes. Este é o campo que nos indica que não se trata do primeiro fragmento, pois o *offset* do primeiro datagrama é sempre 0, visto que é o início do datagrama original.

```
Internet Protocol Version 4, Src: 10.0.2.15, 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: 0xdcd0 (56528)
  Flags: 0x20b9, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
  Fragment offset: 1480
```

Figura 18 - Informações segundo datagrama

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

O valor do MTU é 1500 bytes e o tamanho do nosso datagrama original é 3600 bytes. Fazendo a divisão, o é expectável que o datagrama seja dividido em 3 fragmentos. Pelas alienas anteriores verificamos que é isso que acontece.

O esperado para o tamanho dos fragmentos seria: 1º-1500 bytes, 2º-1500 bytes, 3º-600 bytes. No entanto, não é o que se verifica, pois na realidade o último fragmento tem um *total length* de 640 bytes. Isto deve-se ao facto de que em todos os fragementos 20 bytes serem do header, e por isso é conter efetivamente *total length-20* bytes de payload.

Assim sendo, os dois primeiros fragmentos contêm 1480 bytes e o último contém 620 bytes, totalizando 2960 bytes, ficando a faltar 620 bytes do payload do datagrama original. Somando a esse valor os 20 bytes do header, obtemos os 640 bytes que são o *total length* do último fragmento.

```
Internet Protocol Version 4, Src: 10.0.2.15, 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: 640
    Identification: 0xcdcd0 (56528)
  Flags: 0x0172
    0... .. = Reserved bit: Not set
    .0... .. = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  Fragment offset: 2960
```

Figura 19 - Informações do último fragmento

1.3. 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 do datagrama original é detetado observando os valores do campo *more fragments*. No caso de este ser *more fragments: not set*, significa que este é o último fragmento.

```
Internet Protocol Version 4, Src: 10.0.2.15, 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: 640
    Identification: 0xcdcd0 (56528)
  Flags: 0x0172
    0... .. = Reserved bit: Not set
    .0... .. = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  Fragment offset: 2960
```

Figura 20 - Informações do último fragmento

De forma a listarmos o último fragmento do primeiro datagrama IP segmentado, utilizamos um filtro para listar todos os últimos fragmentos de todos os datagramas captados. O filtro mostra os pacotes cujo campo *more fragments: not set*, de forma a captar apenas os últimos fragmentos, e o campo *offset* diferente de zero, para garantir que não mostra nenhum pacote que não seja fragmento.

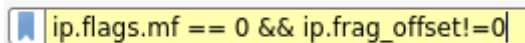
 `ip.flags.mf == 0 && ip.frag_offset!=0`

Figura 21 - Filtro utilizado

A lista resultante é a seguinte:

| No. | Time | Source | Destination | Protocol | Length | Info |
|-----|-------------|-----------|---------------|----------|--------|--|
| 5 | 0.005219... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd0) |
| 8 | 0.007803... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd1) |
| 13 | 0.007982... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd2) |
| 16 | 0.008134... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd3) |
| 20 | 0.008284... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd4) |
| 23 | 0.008445... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd5) |
| 26 | 0.008649... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd6) |
| 29 | 0.008888... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd7) |
| 32 | 0.009119... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd8) |
| 35 | 0.009453... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcd9) |
| 38 | 0.009615... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcdA) |
| 41 | 0.009918... | 10.0.2.15 | 193.136.9.240 | IPv4 | 654 | Fragmented IP protocol (proto=ICMP 1, off=2960, ID=dcdB) |

Figura 22 - Últimos fragmentos do datagrama IP

O último fragmento do primeiro datagrama IP segmentado é primeiro da lista da imagem acima.

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

O datagrama IP original é reconstruído no equipamento destino.

A reconstrução poderia ser feita em outro equipamento, no entanto, não teria nenhuma vantagem em o fazer. Reconstruir o datagrama original em outro local implica todos os fragmentos cheguem ao equipamento em questão para que a reconstrução seja feita, o que impede que os fragmentos possam tomar caminhos diferentes, o que é mais rápido e vantajoso para evitar overloading, para o destino, pois agora todos têm de atingir este nodo intermédio.

Seria também necessário fragmentar novamente o pacote reconstruído para que este chegue destino, o que aumenta desnecessariamente a complexidade e o tempo para atingir o destino.

1.3. 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.

Tal como podemos verificar nas imagens abaixo, os campos no cabeçalho IP que mudam entre os diferentes fragmentos são os seguintes: *total length*, *more fragments* e o *offset*.

```
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.240
0100 .... = Version: 4
... 0101 = Header Length: 20 bytes (5)
Differenziated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0xcdcd0 (56528)
Flags: 0x2000, More fragments
0... .. = Reserved bit: Not set
.0... .. = Don't fragment: Not set
..1... .. = More fragments: Set
Fragment offset: 0

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.240
0100 .... = Version: 4
... 0101 = Header Length: 20 bytes (5)
Differenziated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0xcdcd0 (56528)
Flags: 0x20b9, More fragments
0... .. = Reserved bit: Not set
.0... .. = Don't fragment: Not set
..1... .. = More fragments: Set
Fragment offset: 1480

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.240
0100 .... = Version: 4
... 0101 = Header Length: 20 bytes (5)
Differenziated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 640
Identification: 0xcdcd0 (56528)
Flags: 0x0172
0... .. = Reserved bit: Not set
.0... .. = Don't fragment: Not set
..0... .. = More fragments: Not set
Fragment offset: 2960
```

Figura 23 - Informações dos fragmentos

O campo *fragment offset* indica onde o fragmento atual se encaixa no datagrama original, de forma a poder ordenar-los, aquando da reconstrução do datagrama original. O campo *more fragments* indica se ainda são esperados mais fragmentos ou se esse é o último. O campo *total length* indica o tamanho total do pacote, todos os fragmentos possuem o mesmo exceto o último que pode ter menor visto que é o que “sobra” do datagrama original.

1.3. Alínea h - Por que razão apenas o primeiro fragmento de cada pacote é identificado como sendo um pacote ICMP?

O primeiro fragmento de cada pacote é reconhecido como sendo um ICMP porque contém o cabeçalho do pacote original, incluindo o campo "Protocolo", que indica o tipo de protocolo utilizado no pacote original. Os fragmentos subsequentes não são identificados como pacotes ICMP porque contêm apenas uma parte do payload original e, por isso, não possuem informação para tal identificação. Em vez disso, são simplesmente partes do pacote original, que podem ser reagrupadas pelo destinatário para reconstruir o pacote original.

1.3. 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 tamanho do datagrama é comparado com o valor do MTU (Maximum Transmission Unit), que é o *payload* do pacote Ethernet.

Um aumento do MTU iria diminuir a fragmentação necessária para pacotes grandes, o que também reduziria a sobrecarga nos dispositivos, pois diminui a necessidade de fragmentar e reconstruir. No entanto, aumentaria o tamanho dos pacotes enviados o que poderia implicar uma menor velocidade de transmissão.

Por outro lado, diminuir o valor do MTU diminuiria a confiabilidade da transmissão, pois pacotes menores têm uma maior probabilidade de serem perdidos. No entanto, caso algum erro aconteça, todos os fragmentos são enviados novamente, podendo causar congestionamento na rede.

1.3. 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 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.

O valor máximo de SIZE sem que ocorra fragmentação do pacote é de 1500 bytes, que é o valor do MTU.

O valor máximo de SIZE sem que ocorra fragmentação do pacote é de 1500

O valor máximo que podemos atribuir ao tamanho dos pacotes a ser enviados na execução do comando ping, de forma que não ocorra fragmentação, é 1472 bytes.

Parte 2- Questões e Respostas

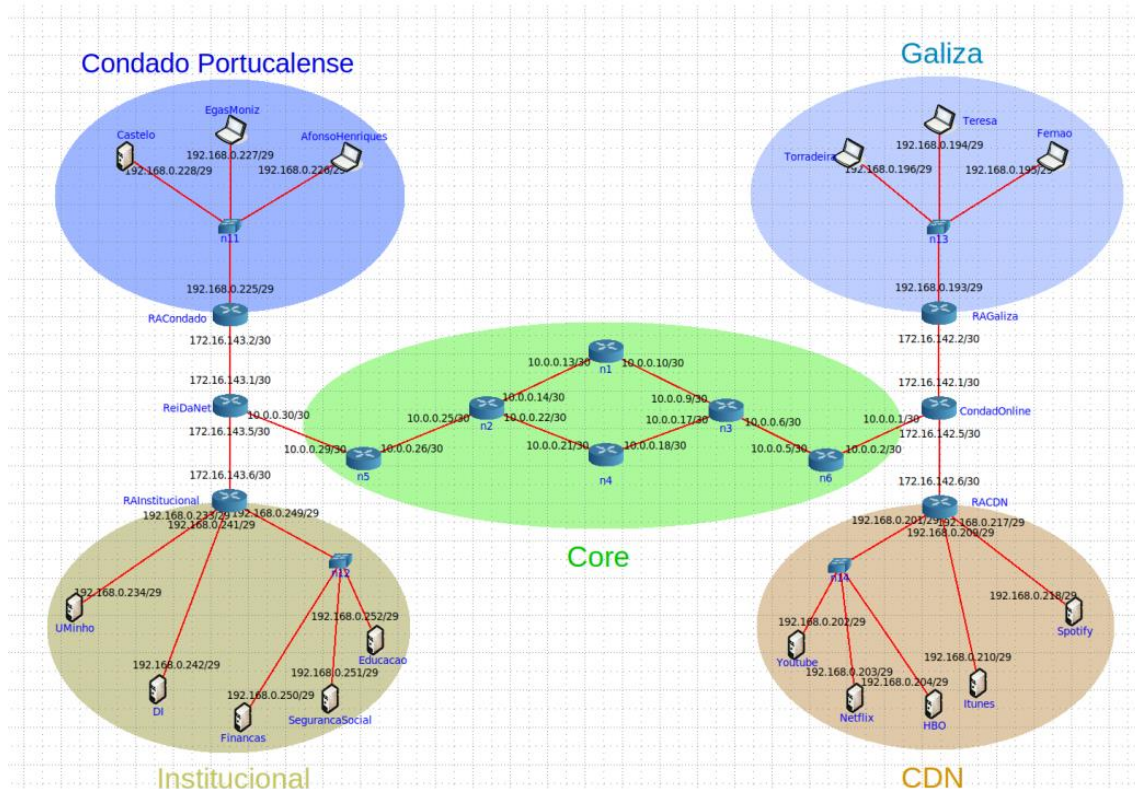


Figura 24 - Topologia Utilizada para a parte 2 do Guião

A topologia é constituída por quatro polos (Condado Portucalense, Galiza, Institucional e CDN (Content Delivery Network)).

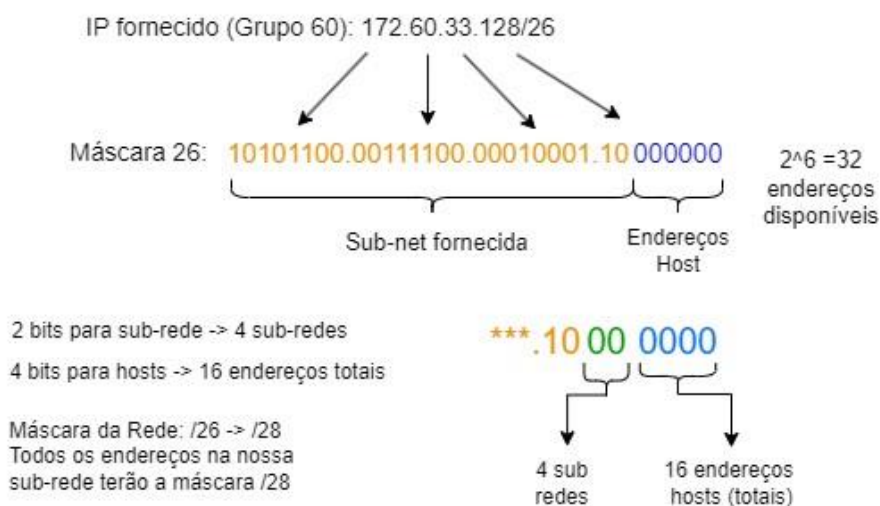
Nos polos Condado e Galiza existe um router de acesso (que permite comunicação com o exterior) ligado a um comutador (switch), por modo a que todos os dispositivos partilhem a mesma rede local. Em cada um dos polos Institucional e CDN encontram-se três sub-redes distintas, criando-se uma segmentação dos dispositivos existentes.

Os polos Condado Portucalense e Institucional estão ligados ao router do ISP ReiDaNet, enquanto Galiza e CDN estão ligados ao ISP CondaOnline. Interligando os dois ISPs existe um ISP de trânsito cuja rede Core é constituída pelos dispositivos n1 a n6.

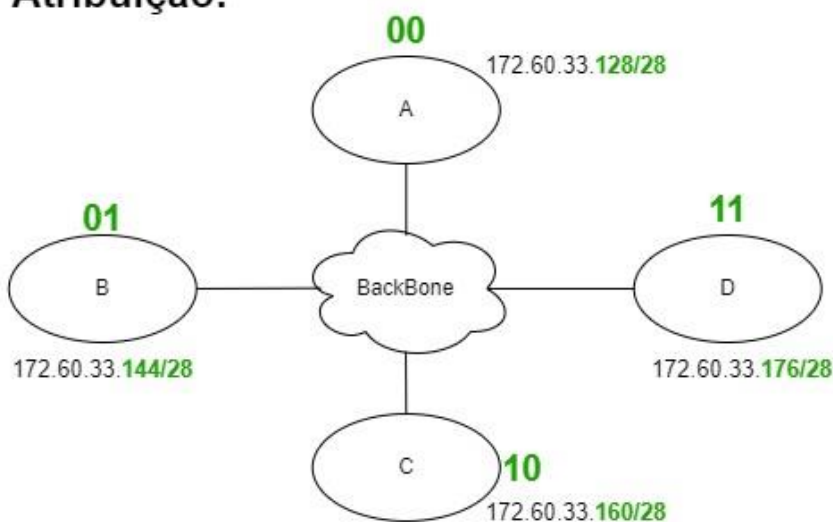
Exercício 1

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

2.1. Alínea a - De modo a garantir uma posição estrategicamente mais vantajosa e ter casa de férias para relaxar entre batalhas, ordena a construção de um segundo Castelo, em Braga. Não tendo qualquer queixa do serviço prestado, recorre aos serviços do ISP ReiDaNet, que já utiliza no condado, para ter acesso à rede no segundo Castelo. O ISP atribuiu-lhe o endereço de rede IP 172.XX.33.128/26 em que XX corresponde ao seu número de grupo (PLXX). Defina um esquema de endereçamento que permita o estabelecimento de pelo menos 3 redes e que garanta que cada uma destas possa ter 12 ou mais hosts. Assuma que todos os endereços de sub-redes são utilizáveis.



Atribuição:



A: de 172.60.33.129/28 até 172.60.33.142/28
B: de 172.60.33.145/28 até 172.60.33.158/28
C: de 172.60.33.161/28 até 172.60.33.174/28
D: de 172.60.33.177/28 até 172.60.33.190/28

O motivo pelo qual escolhemos a máscara /28 deve ao facto de ser pedido para poder ter 3 ou mais redes e 12 ou mais hosts. O número de redes e hosts suportados varia consoante o número de bits que reservamos para cada coisa.

Se fosse escolhida a máscara /27 apenas seríamos capazes de ter 2 redes (1 bit reservado para redes), em vez do mínimo de 3, e se a máscara utilizada fosse /29 apenas poderíamos ter 8 hosts (3 bits reservados para hosts), em vez do mínimo de 12.

2.1. Alínea b - Ligue um novo host Castelo2 diretamente ao router ReiDaNet. Associe-lhe um endereço, à sua escolha, pertencente a uma sub-rede disponível das criadas na alínea anterior (garanta que a interface do router ReiDaNet utiliza o primeiro endereço da sub-rede escolhida). Verifique que tem conectividade com os dispositivos do Condado Portucalense.

Criamos um host e ligamo-lo diretamente ao router ReiDaNet, tal como se pode ver na figura abaixo.

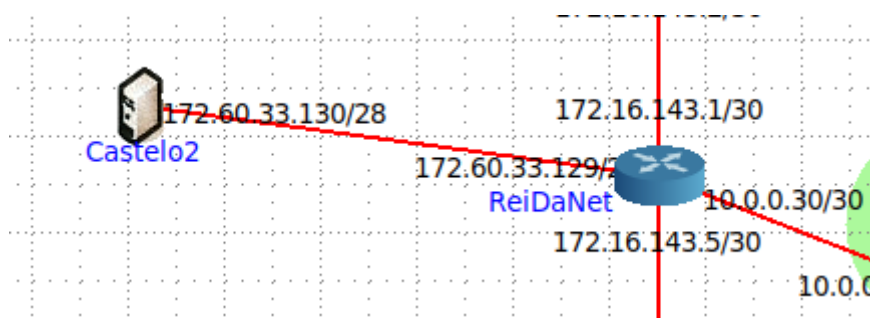


Figura 25 - Novo host Castelo2

Os endereços escolhidos para a ligação ao router e para o host foram, respetivamente, 172.60.33.129/28 e 172.60.33.130/28. Estes dois endereços foram escolhidos da lista de endereços disponíveis na sub-rede criada na alínea anterior, que estão entre 172.60.33.129 e 172.60.33.142.

Para verificar a conectividade com os dispositivos do *Condado Portucalense* realizamos um comando *ping* do Castelo2 para Castelo.

```
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# ping 192.168.0.228
PING 192.168.0.228 (192.168.0.228) 56(84) bytes of data:
64 bytes from 192.168.0.228: icmp_seq=1 ttl=62 time=0.175 ms
64 bytes from 192.168.0.228: icmp_seq=2 ttl=62 time=0.118 ms
64 bytes from 192.168.0.228: icmp_seq=3 ttl=62 time=0.093 ms
^C
--- 192.168.0.228 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2052ms
```

Figura 26 - Ping realizado com sucesso para Castelo

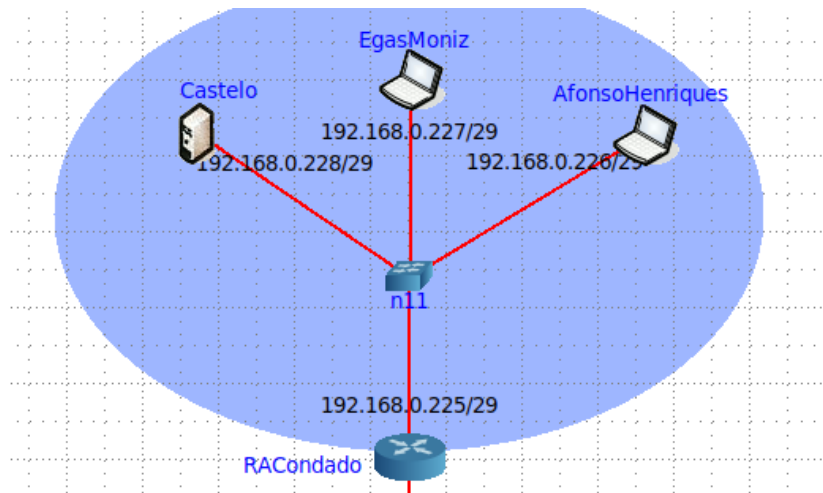


Figura 27 - Localização do host Castelo

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

Através do comando “ip route del default” eliminamos a rota *default*, como se pode verificar na Figura 26. A rota *default* permite que que o dispositivo encaminhe o tráfego para destino fora da sua rede local, caso não esteja especificado uma rota para esse endereço na sua tabela de encaminhamento. Isto reduz o número de rotas da tabela em casos em que vários endereços utilizam a mesma rota.

```
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0        172.60.33.129  0.0.0.0         UG      0 0        0 eth0
172.60.33.128  0.0.0.0        255.255.255.240 U        0 0        0 eth0
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# ip route del default
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
172.60.33.128  0.0.0.0        255.255.255.240 U        0 0        0 eth0
root@Castelo2:/tmp/pycore.35451/Castelo2.conf#
```

Figura 28 - Remoção da rota default

Após remover a rota default, adicionamos uma rota para a rede *Condado Portucalense*, que possui o IP 192.168.0.228.


```

<add -net 192.168.0.224 netmask 255.255.255.248 gw 172.60.33.129
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# ping 192.168.0.228
PING 192.168.0.228 (192.168.0.228) 56(84) bytes of data:
64 bytes from 192.168.0.228: icmp_seq=1 ttl=62 time=0.098 ms
64 bytes from 192.168.0.228: icmp_seq=2 ttl=62 time=0.095 ms
^C
--- 192.168.0.228 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1039ms
rtt min/avg/max/mdev = 0.095/0.096/0.098/0.001 ms
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# netstat -rn
Kernel IP routing table

```

| Destination | Gateway | Genmask | Flags | MSS Window | irrt | Iface |
|---------------|---------------|-----------------|-------|------------|------|-------|
| 172.60.33.128 | 0.0.0.0 | 255.255.255.240 | U | 0 0 | 0 | eth0 |
| 192.168.0.224 | 172.60.33.129 | 255.255.255.248 | UG | 0 0 | 0 | eth0 |

Figura 29 - Adicionada ligação ao Condado Portucalense

```

root@Castelo2:/tmp/pycore.35451/Castelo2.conf# route add -net 192.168.0.232 ne>
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# route add -net 192.168.0.240 ne>
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# route add -net 192.168.0.248 ne>
root@Castelo2:/tmp/pycore.35451/Castelo2.conf# netstat -rn
Kernel IP routing table

```

| Destination | Gateway | Genmask | Flags | MSS Window | irrt | Iface |
|---------------|---------------|-----------------|-------|------------|------|-------|
| 172.60.33.128 | 0.0.0.0 | 255.255.255.240 | U | 0 0 | 0 | eth0 |
| 192.168.0.224 | 172.60.33.129 | 255.255.255.248 | UG | 0 0 | 0 | eth0 |
| 192.168.0.232 | 172.60.33.129 | 255.255.255.248 | UG | 0 0 | 0 | eth0 |
| 192.168.0.240 | 172.60.33.129 | 255.255.255.248 | UG | 0 0 | 0 | eth0 |
| 192.168.0.248 | 172.60.33.129 | 255.255.255.248 | UG | 0 0 | 0 | eth0 |

Figura 30 - Adicionadas as ligações à rede Institucional

No caso deste guião, para obter o endereço da rede era apenas necessário subtrair uma unidade ao último campo do IP do router, o que evitou a necessidade de o calcular.

Exercício 2

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

2.2. Alínea a - Confirme, através do comando ping, que AfonsoHenriques tem efetivamente conectividade com o servidor Finanças e com os servidores da CDN.

Realizamos um *ping*, a partir do host *AfonsoHenriques*, para os servidores *Finanças* e *Itunes* (se atinge *Itunes* então tem conectividade com todos os servidores da *CDN*).

Os resultados dos comandos encontram-se apresentados nas figuras abaixo:

```
<core.35451/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=0.237 ms
64 bytes from 192.168.0.250: icmp_seq=2 ttl=61 time=0.101 ms
^C
--- 192.168.0.250 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1014ms
```

Figura 31 - Ping para o servidor Finanças

```
<core.35451/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=0.403 ms
64 bytes from 192.168.0.210: icmp_seq=2 ttl=55 time=0.253 ms
^C
--- 192.168.0.210 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
```

Figura 32 - Ping para o servidor Itunes

Ambos os *pings* atingiram o seu destino, portanto podemos afirmar que o host *AfonsoHenriques* possui conectividade com o servidor *Finanças* e com todos os servidores da rede *CDN*.

2.2. 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.

Através do comando “*netstat -rn*” aplicado nos terminais de *AfonsoHenriques* e *Teresa*, obtemos as tabelas de encaminhamento dos dispositivos.

```
root@AfonsoHenriques:/tmp/pycore.35451/AfonsoHenriques.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 192.168.0.225 0.0.0.0 UG 0 0 0 eth0
192.168.0.224 0.0.0.0 255.255.255.248 U 0 0 0 eth0
```

Figura 33 - Resultado do comando netstat -rn em AfonsoHenriques

AfonsoHenriques está ligado à interface de IP 192.168.0.225, que é o router da sua rede, assim como confirmado pela tabela de encaminhamento.

```

root@Teresa:/tmp/pycore.35451/Teresa.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0          192.168.0.193  0.0.0.0         UG        0 0          0 eth0
192.168.0.192    0.0.0.0        255.255.255.248 U        0 0          0 eth0

```

Figura 34 - Resultado do comando `netstat -rn` em Teresa

Teresa está ligado à interface de IP 192.168.0.193, que é o router da sua rede, assim como confirmado pela tabela de encaminhamento.

Analisando as suas tabelas podemos verificar que não há nenhum erro com as suas tabelas de encaminhamento, uma vez que, ambas as entidades apontam para o IP da interface do router a que estão conectados.

A primeira linha de ambas as tabelas (*Destination*: 0.0.0.0) é rota *default* do dispositivo. Qualquer endereço que não esteja especificado na tabela será encaminhado para o *gateway* presente na rota *default*. A segunda linha de ambas as tabelas (*Gateway*: 0.0.0.0) indica que todo tráfego interno da rede será feito diretamente, sem a necessidade de um *gateway*.

2.2. Alínea c - Analise 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.

As imagens com a resolução do exercício encontram-se nas próximas páginas.

```

pycore.35451/AfonsoHenriques.conf# traceroute -I 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225)  0.126 ms  0.005 ms  0.003 ms
 2 172.16.143.1 (172.16.143.1)  0.027 ms  0.005 ms  0.005 ms
 3 10.0.0.29 (10.0.0.29)  0.051 ms  !N  0.007 ms  !N *

```

Figura 35 - Tentativa traceroute

```

root@n5:/tmp/pycore.35451/n5.conf# netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags     MSS Window  irtt Iface
10.0.0.0           10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.4           10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.8           10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.12          10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.16          10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.20          10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.24          0.0.0.0           255.255.255.252   U         0 0        0 eth1
10.0.0.28          0.0.0.0           255.255.255.252   U         0 0        0 eth0
172.0.0.0          10.0.0.30         255.0.0.0         UG        0 0        0 eth0
172.16.142.0       10.0.0.25         255.255.255.248   UG        0 0        0 eth1
172.16.143.0       10.0.0.30         255.255.255.252   UG        0 0        0 eth0
172.16.143.0       10.0.0.30         255.255.255.248   UG        0 0        0 eth0
172.16.143.4       10.0.0.30         255.255.255.252   UG        0 0        0 eth0
192.142.0.4        10.0.0.25         255.255.255.252   UG        0 0        0 eth1
192.168.0.200      10.0.0.25         255.255.255.248   UG        0 0        0 eth1
192.168.0.208      10.0.0.25         255.255.255.248   UG        0 0        0 eth1
192.168.0.216      10.0.0.25         255.255.255.248   UG        0 0        0 eth1
192.168.0.224      10.0.0.30         255.255.255.248   UG        0 0        0 eth0
192.168.0.232      10.0.0.30         255.255.255.248   UG        0 0        0 eth0
192.168.0.240      10.0.0.30         255.255.255.248   UG        0 0        0 eth0
192.168.0.248      10.0.0.30         255.255.255.248   UG        0 0        0 eth0

```

Figura 36 - Tabela de endereçamento n5

```

root@n5:/tmp/pycore.35451/n5.conf# route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.25

```

Figura 37 - Comando para adicionar ligação

```

root@n5:/tmp/pycore.35451/n5.conf# netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags     MSS Window  irtt Iface
10.0.0.0           10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.4           10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.8           10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.12          10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.16          10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.20          10.0.0.25         255.255.255.252   UG        0 0        0 eth1
10.0.0.24          0.0.0.0           255.255.255.252   U         0 0        0 eth1
10.0.0.28          0.0.0.0           255.255.255.252   U         0 0        0 eth0
172.0.0.0          10.0.0.30         255.0.0.0         UG        0 0        0 eth0
172.16.142.0       10.0.0.25         255.255.255.248   UG        0 0        0 eth1
172.16.143.0       10.0.0.30         255.255.255.252   UG        0 0        0 eth0
172.16.143.0       10.0.0.30         255.255.255.248   UG        0 0        0 eth0
172.16.143.4       10.0.0.30         255.255.255.252   UG        0 0        0 eth0
192.142.0.4        10.0.0.25         255.255.255.252   UG        0 0        0 eth1
192.168.0.192      10.0.0.25         255.255.255.248   UG        0 0        0 eth1
192.168.0.200      10.0.0.25         255.255.255.248   UG        0 0        0 eth1
192.168.0.208      10.0.0.25         255.255.255.248   UG        0 0        0 eth1
192.168.0.216      10.0.0.25         255.255.255.248   UG        0 0        0 eth1
192.168.0.224      10.0.0.30         255.255.255.248   UG        0 0        0 eth0
192.168.0.232      10.0.0.30         255.255.255.248   UG        0 0        0 eth0
192.168.0.240      10.0.0.30         255.255.255.248   UG        0 0        0 eth0
192.168.0.248      10.0.0.30         255.255.255.248   UG        0 0        0 eth0

```

Figura 38 - Tabela de endereçamento n5 corrigida

A tabela de endereçamento de n5 não possuía uma rota especificada para a rede que pretendemos atingir, e portanto foi necessário adicioná-la.

```

lycore.35451/AfonsoHenriques.conf# traceroute -I 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.046 ms 0.004 ms 0.003 ms
 2 172.16.143.1 (172.16.143.1) 0.016 ms 0.004 ms 0.004 ms
 3 10.0.0.29 (10.0.0.29) 0.021 ms 0.005 ms 0.005 ms
 4 10.0.0.25 (10.0.0.25) 0.043 ms 0.007 ms 0.007 ms^C

```

Figura 39 - Tentativa de traceroute

```

root@n2:/tmp/pycore.35451/n2.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask        Flags   MSS Window  irtt Iface
10.0.0.0         10.0.0.13      255.255.255.252 UG      0 0        0 eth1
10.0.0.4         10.0.0.21      255.255.255.252 UG      0 0        0 eth0
10.0.0.8         10.0.0.13      255.255.255.252 UG      0 0        0 eth1
10.0.0.12        0.0.0.0        255.255.255.252 U       0 0        0 eth1
10.0.0.16        10.0.0.13      255.255.255.252 UG      0 0        0 eth1
10.0.0.20        0.0.0.0        255.255.255.252 U       0 0        0 eth0
10.0.0.24        0.0.0.0        255.255.255.252 U       0 0        0 eth2
10.0.0.28        10.0.0.26      255.255.255.252 UG      0 0        0 eth2
172.0.0.0        10.0.0.26      255.0.0.0      UG      0 0        0 eth2
172.16.142.0     10.0.0.13      255.255.255.252 UG      0 0        0 eth1
172.16.142.4     10.0.0.21      255.255.255.252 UG      0 0        0 eth0
172.16.143.0     10.0.0.26      255.255.255.252 UG      0 0        0 eth2
172.16.143.4     10.0.0.26      255.255.255.252 UG      0 0        0 eth2
192.168.0.192    10.0.0.13      255.255.255.248 UG      0 0        0 eth1
192.168.0.194    10.0.0.25      255.255.255.254 UG      0 0        0 eth2
192.168.0.200    10.0.0.21      255.255.255.248 UG      0 0        0 eth0
192.168.0.208    10.0.0.21      255.255.255.248 UG      0 0        0 eth0
192.168.0.216    10.0.0.21      255.255.255.248 UG      0 0        0 eth0
192.168.0.224    10.0.0.26      255.255.255.248 UG      0 0        0 eth2
192.168.0.232    10.0.0.26      255.255.255.248 UG      0 0        0 eth2
192.168.0.240    10.0.0.26      255.255.255.248 UG      0 0        0 eth2
192.168.0.248    10.0.0.26      255.255.255.248 UG      0 0        0 eth2

```

Figura 40 - Tabela de endereçamento n2

```

root@n2:/tmp/pycore.35451/n2.conf# route del -net 192.168.0.194 netmask 255.255.255.254

```

Figura 41 - Comando para remover ligação

```

root@n2:/tmp/pycore.35451/n2.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask        Flags   MSS Window  irtt Iface
10.0.0.0         10.0.0.13      255.255.255.252 UG      0 0        0 eth1
10.0.0.4         10.0.0.21      255.255.255.252 UG      0 0        0 eth0
10.0.0.8         10.0.0.13      255.255.255.252 UG      0 0        0 eth1
10.0.0.12        0.0.0.0        255.255.255.252 U       0 0        0 eth1
10.0.0.16        10.0.0.13      255.255.255.252 UG      0 0        0 eth1
10.0.0.20        0.0.0.0        255.255.255.252 U       0 0        0 eth0
10.0.0.24        0.0.0.0        255.255.255.252 U       0 0        0 eth2
10.0.0.28        10.0.0.26      255.255.255.252 UG      0 0        0 eth2
172.0.0.0        10.0.0.26      255.0.0.0      UG      0 0        0 eth2
172.16.142.0     10.0.0.13      255.255.255.252 UG      0 0        0 eth1
172.16.142.4     10.0.0.21      255.255.255.252 UG      0 0        0 eth0
172.16.143.0     10.0.0.26      255.255.255.252 UG      0 0        0 eth2
172.16.143.4     10.0.0.26      255.255.255.252 UG      0 0        0 eth2
192.168.0.192    10.0.0.13      255.255.255.248 UG      0 0        0 eth1
192.168.0.200    10.0.0.21      255.255.255.248 UG      0 0        0 eth0
192.168.0.208    10.0.0.21      255.255.255.248 UG      0 0        0 eth0
192.168.0.216    10.0.0.21      255.255.255.248 UG      0 0        0 eth0
192.168.0.224    10.0.0.26      255.255.255.248 UG      0 0        0 eth2
192.168.0.232    10.0.0.26      255.255.255.248 UG      0 0        0 eth2
192.168.0.240    10.0.0.26      255.255.255.248 UG      0 0        0 eth2
192.168.0.248    10.0.0.26      255.255.255.248 UG      0 0        0 eth2

```

Figura 42 - Tabela de endereçamento n2 corrigida

A tabela de endereçamento de n2 possuía uma rota a mais para a rede que queremos atingir, e essa está a ser a rota utilizada pois possui uma máscara mais específica, por isso removemos essa rota.

```
<1/AfonsoHenriques.conf# traceroute -I 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.046 ms 0.004 ms 0.003 ms
 2 172.16.143.1 (172.16.143.1) 0.018 ms 0.005 ms 0.004 ms
 3 10.0.0.29 (10.0.0.29) 0.021 ms 0.005 ms 0.024 ms
 4 10.0.0.25 (10.0.0.25) 0.022 ms 0.006 ms 0.006 ms
 5 10.0.0.13 (10.0.0.13) 0.047 ms 0.008 ms 0.007 ms
 6 10.0.0.25 (10.0.0.25) 0.007 ms 0.026 ms 0.008 ms
 7 10.0.0.13 (10.0.0.13) 0.008 ms 0.008 ms 0.008 ms^C
```

Figura 43 - Tentativa de traceroute

```
root@n1:/tmp/pycore.35451/n1.conf# netstat -rn
Kernel IP routing table
Destination        Gateway           Genmask          Flags        MSS Window  irtt  Iface
10.0.0.0           10.0.0.9         255.255.255.252 UG           0 0        0     eth0
10.0.0.4           10.0.0.9         255.255.255.252 UG           0 0        0     eth0
10.0.0.8           0.0.0.0          255.255.255.252 U            0 0        0     eth0
10.0.0.12          0.0.0.0          255.255.255.252 U            0 0        0     eth1
10.0.0.16          10.0.0.9         255.255.255.252 UG           0 0        0     eth0
10.0.0.20          10.0.0.14        255.255.255.252 UG           0 0        0     eth1
10.0.0.24          10.0.0.14        255.255.255.252 UG           0 0        0     eth1
10.0.0.28          10.0.0.14        255.255.255.252 UG           0 0        0     eth1
172.0.0.0          10.0.0.14        255.0.0.0       UG           0 0        0     eth1
172.16.142.0       10.0.0.9         255.255.255.252 UG           0 0        0     eth0
172.16.142.4       10.0.0.9         255.255.255.252 UG           0 0        0     eth0
172.16.143.0       10.0.0.14        255.255.255.252 UG           0 0        0     eth1
172.16.143.4       10.0.0.14        255.255.255.252 UG           0 0        0     eth1
192.168.0.192      10.0.0.14        255.255.255.248 UG           0 0        0     eth1
192.168.0.200      10.0.0.9         255.255.255.248 UG           0 0        0     eth0
192.168.0.208      10.0.0.9         255.255.255.248 UG           0 0        0     eth0
192.168.0.216      10.0.0.9         255.255.255.248 UG           0 0        0     eth0
192.168.0.224      10.0.0.14        255.255.255.248 UG           0 0        0     eth1
192.168.0.232      10.0.0.14        255.255.255.248 UG           0 0        0     eth1
192.168.0.240      10.0.0.14        255.255.255.248 UG           0 0        0     eth1
192.168.0.248      10.0.0.14        255.255.255.248 UG           0 0        0     eth1
```

Figura 44 - Tabela de endereçamento n1

```
root@n1:/tmp/pycore.35451/n1.conf# route del -net 192.168.0.192 netmask 255.255.255.248
```

Figura 45 - Comando para remover ligação

```
root@n1:/tmp/pycore.35451/n1.conf# route add -net 192.168.0.192 netmask 255.255.255.248 gw 10.0.0.9
```

Figura 46 - Comando para adicionar ligação

```
<1/AfonsoHenriques.conf# traceroute -I 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.048 ms 0.005 ms 0.003 ms
 2 172.16.143.1 (172.16.143.1) 0.018 ms 0.004 ms 0.004 ms
 3 10.0.0.29 (10.0.0.29) 0.021 ms 0.005 ms 0.005 ms
 4 10.0.0.25 (10.0.0.25) 0.022 ms 0.007 ms 0.006 ms
 5 10.0.0.13 (10.0.0.13) 0.022 ms 0.008 ms 0.008 ms
 6 10.0.0.17 (10.0.0.17) 0.081 ms 0.022 ms 0.009 ms
 7 10.0.0.5 (10.0.0.5) 0.072 ms 0.014 ms 0.010 ms
 8 10.0.0.1 (10.0.0.1) 0.065 ms 0.014 ms 0.011 ms^C
```

Figura 47 - Traceroute bem-sucedido para CondaOnline

Na tabela de endereçamento de n1 a rota para a rede que queremos atingir possui o gateway errado, e por isso esta é removida e adicionamos uma nova rota, para a mesma rede, mas com o gateway correto.

Após todas estas correções, verificamos na Figura 45 que o traceroute já atinge o destino.

2.2. 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.

2.2.d.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.

| | | | | | | |
|----|-------------|-------------------|---------------|------|---|---|
| 17 | 26.62942... | 192.168.0.226 | 192.168.0.194 | ICMP | 98 Echo (ping) request | id=0x0043, seq=1/256, ttl=55 (reply in 18) |
| 18 | 26.62944... | 192.168.0.194 | 192.168.0.226 | ICMP | 98 Echo (ping) reply | id=0x0043, seq=1/256, ttl=64 (request in 17) |
| 19 | 26.62945... | 192.168.0.193 | 192.168.0.194 | ICMP | 126 Destination unreachable (Network unreachable) | |
| 20 | 27.66094... | 192.168.0.226 | 192.168.0.194 | ICMP | 98 Echo (ping) request | id=0x0043, seq=2/512, ttl=55 (reply in 21) |
| 21 | 27.66097... | 192.168.0.194 | 192.168.0.226 | ICMP | 98 Echo (ping) reply | id=0x0043, seq=2/512, ttl=64 (request in 20) |
| 22 | 27.66098... | 192.168.0.193 | 192.168.0.194 | ICMP | 126 Destination unreachable (Network unreachable) | |
| 23 | 28.02156... | 192.168.0.193 | 224.0.0.5 | OSPF | 78 Hello Packet | |
| 24 | 28.67959... | fe80::200:ff:f... | ff02::5 | OSPF | 90 Hello Packet | |
| 25 | 28.68451... | 192.168.0.226 | 192.168.0.194 | ICMP | 98 Echo (ping) request | id=0x0043, seq=3/768, ttl=55 (reply in 26) |
| 26 | 28.68452... | 192.168.0.194 | 192.168.0.226 | ICMP | 98 Echo (ping) reply | id=0x0043, seq=3/768, ttl=64 (request in 25) |
| 27 | 28.68453... | 192.168.0.193 | 192.168.0.194 | ICMP | 126 Destination unreachable (Network unreachable) | |
| 28 | 29.71000... | 192.168.0.226 | 192.168.0.194 | ICMP | 98 Echo (ping) request | id=0x0043, seq=4/1024, ttl=55 (reply in 29) |
| 29 | 29.71001... | 192.168.0.194 | 192.168.0.226 | ICMP | 98 Echo (ping) reply | id=0x0043, seq=4/1024, ttl=64 (request in 28) |
| 30 | 29.71003... | 192.168.0.193 | 192.168.0.194 | ICMP | 126 Destination unreachable (Network unreachable) | |
| 31 | 30.02245... | 192.168.0.193 | 224.0.0.5 | OSPF | 78 Hello Packet | |
| 32 | 30.73452... | 192.168.0.226 | 192.168.0.194 | ICMP | 98 Echo (ping) request | id=0x0043, seq=5/1280, ttl=55 (reply in 33) |
| 33 | 30.73454... | 192.168.0.194 | 192.168.0.226 | ICMP | 98 Echo (ping) reply | id=0x0043, seq=5/1280, ttl=64 (request in 32) |

Figura 48 - Captura Wireshark Teresa

Como podemos verificar pela figura acima, os pacotes chegam a *Teresa*. É recebido o *request*, enviado o *reply*, e obtemos um erro que indica que não foi possível atingir o destino. Através da *Figura 47*, a execução de um comando *traceroute* de *Teresa* para *AfonsoHenriques*, verificamos que não só não atinge o destino como apenas atinge o router da rede e nada mais após isso.

```
root@Teresa:/tmp/pycore.35451/Teresa.conf# traceroute -I 192.168.0.227
traceroute to 192.168.0.227 (192.168.0.227), 30 hops max, 60 byte packets
 1 192.168.0.193 (192.168.0.193) 0.035 ms !N 0.004 ms !N *
```

Figura 49 - Tentativa de traceroute

| root@RAGaliza:/tmp/pycore.35451/RAGaliza.conf# netstat -rn | | | | | | | |
|--|--------------|-----------------|-------|-----|--------|------|-------|
| Kernel IP routing table | | | | | | | |
| Destination | Gateway | Genmask | Flags | MSS | Window | irtt | Iface |
| 10.0.0.0 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 10.0.0.4 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 10.0.0.8 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 10.0.0.12 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 10.0.0.16 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 10.0.0.20 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 10.0.0.24 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 10.0.0.28 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 172.0.0.0 | 172.16.142.1 | 255.0.0.0 | UG | 0 0 | | 0 | eth0 |
| 172.16.142.0 | 0.0.0.0 | 255.255.255.252 | U | 0 0 | | 0 | eth0 |
| 172.16.142.4 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 172.16.143.0 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 172.16.143.4 | 172.16.142.1 | 255.255.255.252 | UG | 0 0 | | 0 | eth0 |
| 192.168.0.192 | 0.0.0.0 | 255.255.255.248 | U | 0 0 | | 0 | eth1 |
| 192.168.0.200 | 172.16.142.1 | 255.255.255.248 | UG | 0 0 | | 0 | eth0 |
| 192.168.0.208 | 172.16.142.1 | 255.255.255.248 | UG | 0 0 | | 0 | eth0 |
| 192.168.0.216 | 172.16.142.1 | 255.255.255.248 | UG | 0 0 | | 0 | eth0 |
| 192.168.0.232 | 172.16.142.1 | 255.255.255.248 | UG | 0 0 | | 0 | eth0 |
| 192.168.0.240 | 172.16.142.1 | 255.255.255.248 | UG | 0 0 | | 0 | eth0 |
| 192.168.0.248 | 172.16.142.1 | 255.255.255.248 | UG | 0 0 | | 0 | eth0 |

Figura 50 - Tabela de endereçamento RAGaliza

Como podemos verificar pela tabela de endereçamento do router *RAGaliza*, não existe uma rota para a rede pretendida, por isso esta é adicionada, usando o comando da *Figura 49*.

```
root@RAGaliza:/tmp/pycore.35451/RAGaliza.conf# route add -net 192.168.0.224 netmask 255.255.255.248 gw 172.16.142.1
```

Figura 51 - Comando para adicionar ligação

```
root@RAGaliza:/tmp/pycore.35451/RAGaliza.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt Iface
10.0.0.0         172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.4         172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.8         172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.12        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.16        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.20        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.24        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.28        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
172.0.0.0        172.16.142.1    255.0.0.0       UG        0 0          0 eth0
172.16.142.0     0.0.0.0         255.255.255.252 U        0 0          0 eth0
172.16.142.4     172.16.142.1    255.255.255.252 UG        0 0          0 eth0
172.16.143.0     172.16.142.1    255.255.255.252 UG        0 0          0 eth0
172.16.143.4     172.16.142.1    255.255.255.252 UG        0 0          0 eth0
192.168.0.192    0.0.0.0         255.255.255.248 U        0 0          0 eth1
192.168.0.200    172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.208    172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.216    172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.224    172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.232    172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.240    172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.248    172.16.142.1    255.255.255.248 UG        0 0          0 eth0
```

Figura 52 - Tabela de endereçamento *RAGaliza* corrigida

```
root@Teresa:/tmp/pycore.35451/Teresa.conf# traceroute -I 192.168.0.227
traceroute to 192.168.0.227 (192.168.0.227), 30 hops max, 60 byte packet
 1 192.168.0.193 (192.168.0.193) 0.058 ms 0.004 ms 0.004 ms
 2 172.16.142.1 (172.16.142.1) 0.048 ms 0.005 ms 0.004 ms
 3 10.0.0.2 (10.0.0.2) 0.021 ms 0.006 ms 0.005 ms
 4 10.0.0.6 (10.0.0.6) 0.053 ms 0.008 ms 0.006 ms
 5 10.0.0.18 (10.0.0.18) 0.050 ms 0.009 ms 0.008 ms
 6 10.0.0.14 (10.0.0.14) 0.061 ms 0.054 ms 0.010 ms
 7 10.0.0.26 (10.0.0.26) 0.068 ms 0.014 ms 0.010 ms
 8 10.0.0.30 (10.0.0.30) 0.024 ms 0.012 ms 0.013 ms
 9 172.16.143.2 (172.16.143.2) 0.052 ms 0.016 ms 0.013 ms
10 192.168.0.227 (192.168.0.227) 0.076 ms 0.020 ms 0.014 ms
```

Figura 53 - Traceroute bem-sucedido para "*AfonsoHenriques*"

| | | | | | |
|----------------|---------------|---------------|------|------------------------|--|
| 10 15.59002... | 192.168.0.226 | 192.168.0.194 | ICMP | 98 Echo (ping) request | id=0x004c, seq=1/256, ttl=55 (reply in 11) |
| 11 15.59003... | 192.168.0.194 | 192.168.0.226 | ICMP | 98 Echo (ping) reply | id=0x004c, seq=1/256, ttl=64 (request in 10) |
| 12 16.01726... | 192.168.0.193 | 224.0.0.5 | OSPF | 78 Hello Packet | |
| 13 16.52569... | 192.168.0.226 | 192.168.0.194 | ICMP | 98 Echo (ping) request | id=0x004c, seq=2/512, ttl=55 (reply in 14) |
| 14 16.52571... | 192.168.0.194 | 192.168.0.226 | ICMP | 98 Echo (ping) reply | id=0x004c, seq=2/512, ttl=64 (request in 13) |

Figura 54 -Excerto WhireShark que comprova que foi bem sucedido

Após a correção da tabela de endereçamento do router *RAGaliza* verificamos que o comando *traceroute* já atinge o destino.

2.2.d.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.

Como podemos verificar pelas figuras *Figura 53* e *Figura 54* a rota que os pacotes *ICMP echo request* e *ICMP echo reply* utilizam na comunicação entre *AfonsoHenriques* e *Teresa* não são as mesmas. Nas bifurcações que ocorrem nos routers n2 e n3 os pacotes tomam caminho diferentes, um “por cima” e outro “por baixo”.

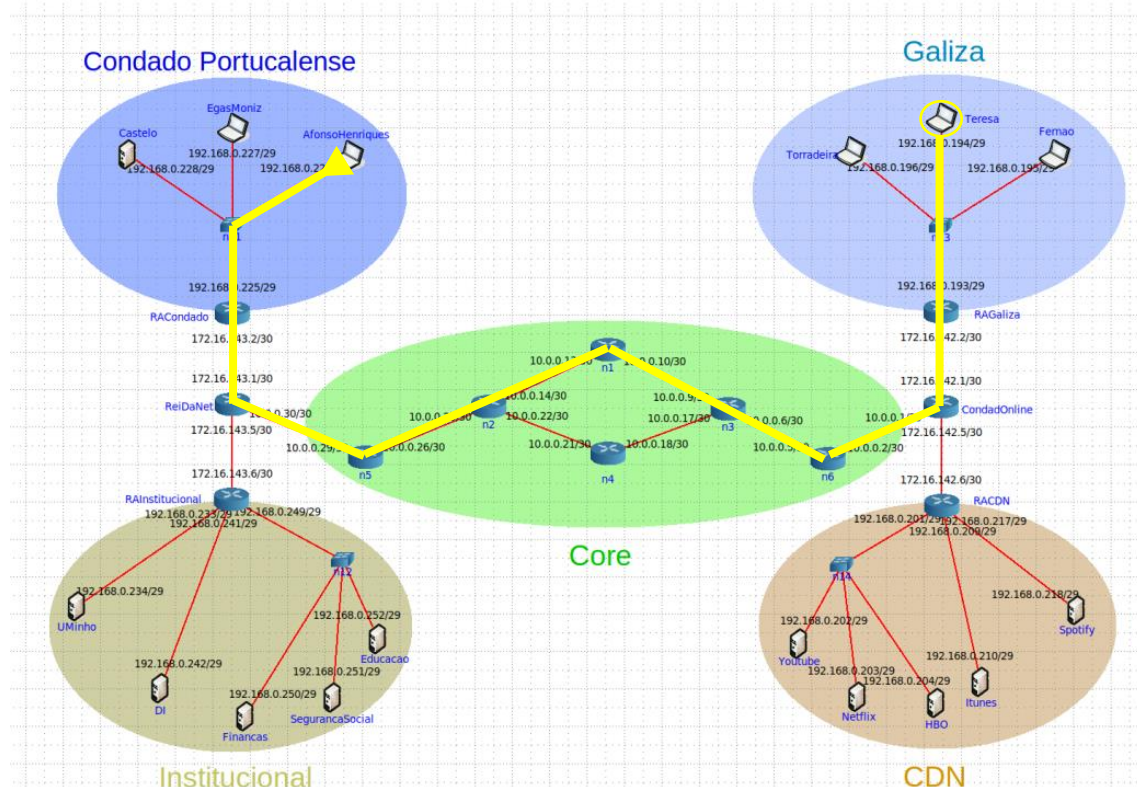


Figura 55 - Rota entre "AfonsoHenriques" e "Teresa"

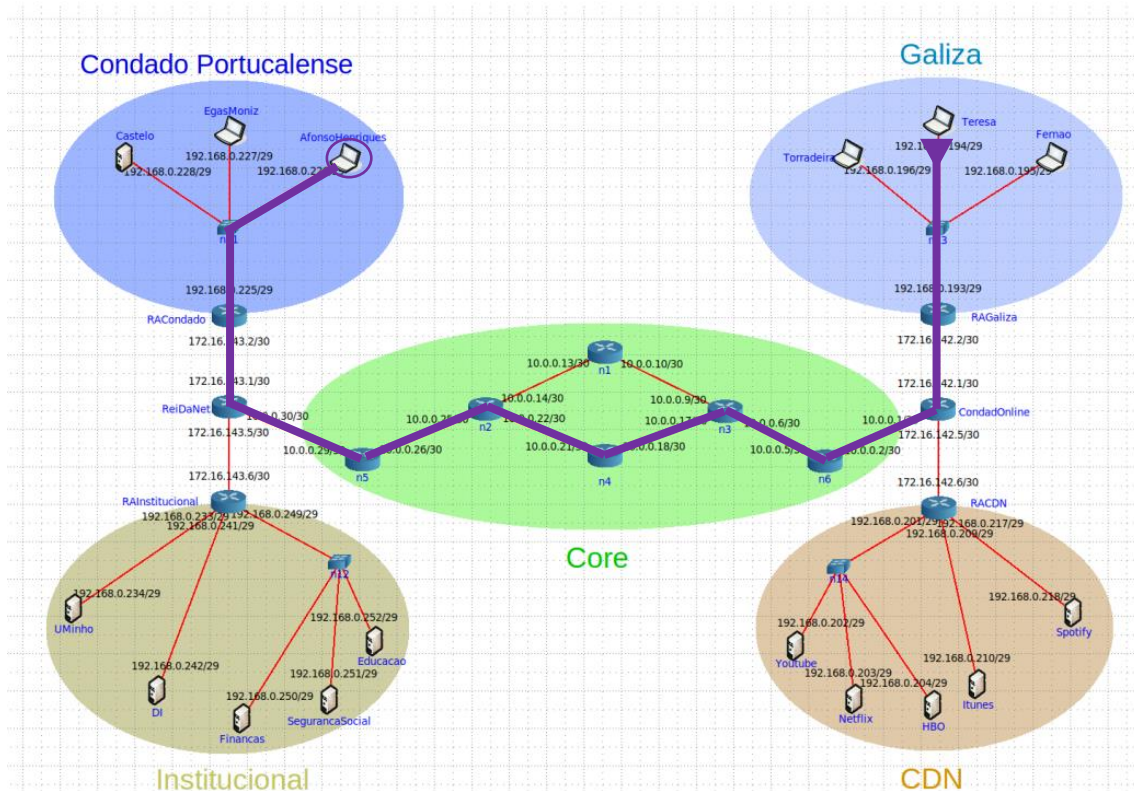


Figura 56 - Rota entre "Teresa" e "AfonsoHenriques"

```
root@Teresa:/tmp/pycore.35451/Teresa.conf# traceroute -I 192.168.0.227
traceroute to 192.168.0.227 (192.168.0.227), 30 hops max, 60 byte packets
 1 192.168.0.193 (192.168.0.193) 0.035 ms 0.004 ms 0.003 ms
 2 172.16.142.1 (172.16.142.1) 0.018 ms 0.004 ms 0.004 ms
 3 10.0.0.2 (10.0.0.2) 0.017 ms 0.005 ms 0.005 ms
 4 10.0.0.6 (10.0.0.6) 0.016 ms 0.006 ms 0.006 ms
 5 10.0.0.18 (10.0.0.18) 0.018 ms 0.008 ms 0.007 ms
 6 10.0.0.14 (10.0.0.14) 0.026 ms 0.022 ms 0.009 ms
 7 10.0.0.26 (10.0.0.26) 0.019 ms 0.010 ms 0.010 ms
 8 10.0.0.30 (10.0.0.30) 0.019 ms 0.011 ms 0.011 ms
 9 172.16.143.2 (172.16.143.2) 0.021 ms 0.012 ms 0.012 ms
10 192.168.0.227 (192.168.0.227) 0.023 ms 0.013 ms 0.013 ms
```

Figura 57 - Rotas pacotes ICMP entre "AfonsoHenriques" e "Teresa"

```
<core.35451/AfonsoHenriques.conf# traceroute -I 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.034 ms 0.004 ms 0.003 ms
 2 172.16.143.1 (172.16.143.1) 0.018 ms 0.005 ms 0.004 ms
 3 10.0.0.29 (10.0.0.29) 0.020 ms 0.006 ms 0.006 ms
 4 10.0.0.25 (10.0.0.25) 0.019 ms 0.006 ms 0.006 ms
 5 10.0.0.13 (10.0.0.13) 0.021 ms 0.007 ms 0.007 ms
 6 10.0.0.17 (10.0.0.17) 0.036 ms 0.032 ms 0.009 ms
 7 10.0.0.5 (10.0.0.5) 0.032 ms 0.010 ms 0.010 ms
 8 10.0.0.1 (10.0.0.1) 0.025 ms 0.011 ms 0.010 ms
 9 172.16.142.2 (172.16.142.2) 0.023 ms 0.013 ms 0.013 ms
10 192.168.0.194 (192.168.0.194) 0.023 ms 0.014 ms 0.013 ms
```

Figura 58 - Rotas pacotes ICMP entre "Teresa" e "AfonsoHenriques"

2.2. 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 10.0.0.18 255.255.255.240 UG 0 0 0 eth1
192.168.0.192 10.0.0.18 255.255.255.240 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.

Existe match para os pacotes enviados para o polo *Galiza*, pois essa linha especifica a rota a ser utilizada para a rede *Galiza*. No entanto, essa rota envia o pacote para o router n4 (Interface com IP: 10.0.0.18) e a rota especificada nesse router para a rede Galiza é de volta para o n1. Ou seja, o pacote entra num ciclo em que do n1 para o n4 e do n4 para o n1 indefinidamente. Assim sendo, este nunca atingirá a rede Galiza.

O mesmo se aplica para o polo *CDN*, irá entrar em loop e num atingir o destino pretendido.

```
root@n4:/tmp/pycore.35451/n4.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 10.0.0.17 255.255.255.252 UG 0 0 0 eth0
10.0.0.4 10.0.0.17 255.255.255.252 UG 0 0 0 eth0
10.0.0.8 10.0.0.17 255.255.255.252 UG 0 0 0 eth0
10.0.0.12 10.0.0.22 255.255.255.252 UG 0 0 0 eth1
10.0.0.16 0.0.0.0 255.255.255.252 U 0 0 0 eth0
10.0.0.20 0.0.0.0 255.255.255.252 U 0 0 0 eth1
10.0.0.24 10.0.0.22 255.255.255.252 UG 0 0 0 eth1
10.0.0.28 10.0.0.22 255.255.255.252 UG 0 0 0 eth1
172.0.0.0 10.0.0.22 255.0.0.0 UG 0 0 0 eth1
172.16.142.0 10.0.0.17 255.255.255.252 UG 0 0 0 eth0
172.16.142.4 10.0.0.17 255.255.255.252 UG 0 0 0 eth0
172.16.143.0 10.0.0.22 255.255.255.252 UG 0 0 0 eth1
172.16.143.4 10.0.0.22 255.255.255.252 UG 0 0 0 eth1
192.168.0.192 10.0.0.17 255.255.255.248 UG 0 0 0 eth0
192.168.0.200 10.0.0.17 255.255.255.248 UG 0 0 0 eth0
192.168.0.208 10.0.0.17 255.255.255.248 UG 0 0 0 eth0
192.168.0.216 10.0.0.17 255.255.255.248 UG 0 0 0 eth0
192.168.0.224 10.0.0.22 255.255.255.248 UG 0 0 0 eth1
192.168.0.232 10.0.0.22 255.255.255.248 UG 0 0 0 eth1
192.168.0.240 10.0.0.22 255.255.255.248 UG 0 0 0 eth1
192.168.0.248 10.0.0.22 255.255.255.248 UG 0 0 0 eth1
```

Figura 59 - Tabela de endereçamento de n1

2.2. 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 usados em toda a topologia são endereços privados, visto que se encontram dentro dos seguintes intervalos de endereços:

- 10.0.0.0 a 10.255.255.255
- 172.16.0.0 a 172.31.255.255
- 192.168.0.0 a 192.168.255.255

Esses endereços foram especificamente reservados pela Internet Assigned Numbers Authority (IANA) para uso em redes locais privadas, o que permite que as organizações utilizem sem o risco de conflito com dispositivos na Internet.

2.2. Alínea g - Os switches localizados em cada um dos polos têm um endereço IP atribuído? Porquê?

Os *switches* não contêm endereços IP visto que são componentes de nível inferior na topologia de rede. Estes *switches* apenas auxiliam no envio dos pacotes aos recipientes/equipamentos corretos através dos endereços MAC, não sendo necessário atribuir um endereço de rede, pois os *switches* funciona ao nível de ligação de dados.

Exercício 3

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

2.3. Alínea A e Alínea B

Pergunta 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.

Após eliminar as rotas com destino a qualquer IP na “Galiza” ou “CDN”, aplicamos o comando “ip route add 192.168.0.192/27 via 10.0.0.1” e conseguimos assim criar um Supernetting que conecta os dois polos.

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

Apagamos todas as rotas com destino no “Condado Portucalense” e “Institucional”. Através do comando “ip route add 192.168.0.224/27 via 10.0.0.6” criamos outras Supernet agora para estes polos.

Como ambas as alíneas pedem algo semelhante juntamos a resposta numa só.

Para fazermos o supernetting das rotas devemos apagar as entradas individuais para cada uma das redes e posteriormente adicionar uma única entrada que agrupe as entradas originais. Para criar uma entrada que consiga agrupar as entradas originais devemos começar por passar os IPs de cada entrada para binário e verificar qual o maior prefixo de bits comuns.

No caso das entradas tanto para a alínea a) e para a alínea b) vemos que os primeiros 27 bits são comuns, logo a máscara de rede será /27.

192.168.0.192 -> 11000000.10101000.00000000.11000000

192.168.0.200 -> 11000000.10101000.00000000.11001000

192.168.0.208 -> 11000000.10101000.00000000.11010000

192.168.0.216 -> 11000000.10101000.00000000.11011000

De seguida devemos verificar qual o IP dessa supernetting que receberá a máscara /27, novamente analisando os valores em binário. Como os 3 primeiros octetos são iguais e só os 3 primeiros bits do último octeto são comuns, então os últimos 5 bits são reduzidos a 0s, logo o IP que agrupa estas entradas é 192.168.0.192/27 (Alínea a) e 192.168.0.224/27 (alínea b)). De seguida, escolhemos a Gateway pela qual chegarão a essas redes que é a Gateway original de cada uma das entradas (10.0.0.1 para a alínea a) e 10.0.0.6 para alínea b)) logo através dos comandos – “ip route add 192.168.0.192/27 via 10.0.0.1” e “ ip route add 192.168.0.224/27 via 10.0.0.6” adicionamos as entradas para as supernets da alínea a) e b) respetivamente.

```
10.0.0.16 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.20 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.24 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.28 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
172.0.0.0 10.0.0.6 255.0.0.0 UG 0 0 0 eth1
172.16.142.0 10.0.0.1 255.255.255.252 UG 0 0 0 eth0
172.16.142.4 10.0.0.1 255.255.255.252 UG 0 0 0 eth0
172.16.143.0 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
172.16.143.4 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
192.168.0.192 10.0.0.1 255.255.255.248 UG 0 0 0 eth0
192.168.0.200 10.0.0.1 255.255.255.248 UG 0 0 0 eth0
192.168.0.208 10.0.0.1 255.255.255.248 UG 0 0 0 eth0
192.168.0.216 10.0.0.1 255.255.255.248 UG 0 0 0 eth0
192.168.0.224 10.0.0.6 255.255.255.248 UG 0 0 0 eth1
192.168.0.232 10.0.0.6 255.255.255.248 UG 0 0 0 eth1
192.168.0.240 10.0.0.6 255.255.255.248 UG 0 0 0 eth1
192.168.0.248 10.0.0.6 255.255.255.248 UG 0 0 0 eth1
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.192/29 via 10.0.0.1
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.200/29 via 10.0.0.1
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.208/29 via 10.0.0.1
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.216/29 via 10.0.0.1
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.224/29 via 10.0.0.6
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.232/29 via 10.0.0.6
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.240/29 via 10.0.0.6
root@61:/tmp/pscore.37667/n6.conf# ip route del 192.168.0.248/29 via 10.0.0.6
root@61:/tmp/pscore.37667/n6.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 0.0.0.0 255.255.255.252 U 0 0 0 eth0
10.0.0.4 0.0.0.0 255.255.255.252 U 0 0 0 eth1
10.0.0.8 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.12 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.16 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.20 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.24 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.28 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
172.0.0.0 10.0.0.6 255.0.0.0 UG 0 0 0 eth1
172.16.142.0 10.0.0.1 255.255.255.252 UG 0 0 0 eth0
172.16.142.4 10.0.0.1 255.255.255.252 UG 0 0 0 eth0
172.16.143.0 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
172.16.143.4 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
root@61:/tmp/pscore.37667/n6.conf# ip route add 192.168.0.110/27 via 10.0.0.1
Error: Invalid prefix for given prefix length.
root@61:/tmp/pscore.37667/n6.conf# ip route add 192.168.0.192/27 via 10.0.0.1
root@61:/tmp/pscore.37667/n6.conf# ip route add 192.168.0.224/27 via 10.0.0.6
root@61:/tmp/pscore.37667/n6.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 0.0.0.0 255.255.255.252 U 0 0 0 eth0
10.0.0.4 0.0.0.0 255.255.255.252 U 0 0 0 eth1
10.0.0.8 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.12 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.16 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.20 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.24 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
10.0.0.28 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
172.0.0.0 10.0.0.6 255.0.0.0 UG 0 0 0 eth1
172.16.142.0 10.0.0.1 255.255.255.252 UG 0 0 0 eth0
172.16.142.4 10.0.0.1 255.255.255.252 UG 0 0 0 eth0
172.16.143.0 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
172.16.143.4 10.0.0.6 255.255.255.252 UG 0 0 0 eth1
192.168.0.192 10.0.0.1 255.255.255.224 UG 0 0 0 eth0
192.168.0.224 10.0.0.6 255.255.255.224 UG 0 0 0 eth1
```

2.3. Alínea c - Comente os aspetos positivos e negativos do uso do Supernetting.

Supernetting consiste na agregação de várias sub-redes adjacentes em uma única rede maior, graças ao facto de as máscaras poderem ter qualquer valor (devido ao endereçamento sem classes). Esta técnica é geralmente usada para reduzir o número de entradas nas tabelas de encaminhamento dos routers e melhorar a eficiência da rede.

O uso de *Supernetting* tem vários pontos positivos, bem como pontos negativos.

Focando-nos, inicialmente, nos pontos positivos da utilização de *Supernetting*, uma das grandes vantagens que esta técnica oferece é a redução do número de entradas nas tabelas de encaminhamento dos routers. Isto permite um melhor desempenho da rede pois, uma menor quantidade de entradas nas tabelas de encaminhamento significa que há menos informação para os routers processarem, diminuindo, assim, sobrecarga de processamento.

A utilização de *Supernetting* pode, ainda, ajudar a conservar endereços IP pois reduz o número de sub-redes que precisam ser alocadas, permitindo que mais dispositivos sejam conectados à rede.

No entanto, o *Supernetting* pode ser inflexível, pois todas as sub-redes agregadas devem ter o mesmo tamanho de prefixo, o que pode dificultar a adição de novas sub-redes.

Adicionalmente, o *Supernetting* pode aumentar o risco de congestão de rede, especialmente se a rede estiver sobrecarregada ou houver um grande número de dispositivos conectados à rede. Se a rede ficar congestionada, pode haver atrasos ou perda de pacotes, o que pode afetar a qualidade do serviço fornecido aos usuários.

Conclusão

Realizar este trabalho possibilitou um aprofundamento considerável dos conhecimentos adquiridos nas aulas teóricas. Foi-nos possível colocar em prática o funcionamento do comando traceroute, realizar análises de pacotes IP e compreender o processo de fragmentação dos mesmos. Além disso, aprendeu-se a identificar os procedimentos necessários para verificar o funcionamento adequado de uma rede local, consultar tabelas de encaminhamento, as implicações de remover rotas (como a rota default) e, ainda, como corrigir possíveis erros no encaminhamento.