



UNIVERSIDADE DO MINHO
LICENCIATURA EM ENGENHARIA INFORMÁTICA

REDES DE COMPUTADORES

Trabalho Prático 2 PROTOCOLO IPv4: DATAGRAMAS IP E FRAGMENTAÇÃO

Mariana Gonçalves (a100662) Maya Gomes (a100822) Vicente Martins
(a100713)

Conteúdos

1	Introdução	4
2	Part.I do Trabalho prático	4
2.1	Questões e Respostas I	5
2.1.1	Questão A	5
2.1.2	Questão B	7
2.1.3	Questão C	8
2.1.4	Questão D	8
2.2	Questões e respostas II	9
2.2.1	Questão A	9
2.2.2	Questão B	9
2.2.3	Questão C	9
2.2.4	Questão D	10
2.2.5	Questão E	10
2.2.6	Questão F	11
2.2.7	Questão G	13
2.2.8	Questão H	14
2.3	Questões e respostas III	15
2.3.1	Questão A	15
2.3.2	Questão B	16
2.3.3	Questão C	17
2.3.4	Questão D	17
2.3.5	Questão E	18
2.3.6	Questão F	18
2.3.7	Questão G	19
2.3.8	Questão H	19
2.3.9	Questão I	20
2.3.10	Questão J	20
3	Part.II do Trabalho prático	21
3.1	Questões e Respostas I	21
3.1.1	Questão A	21
3.1.2	Questão B	22
3.1.3	Questão C	24
3.1.4	Questão D	28
3.1.5	Questão E	30
3.1.6	Questão F	32
3.1.7	Questão G	32
3.2	Questões e Respostas II	33
3.2.1	Questão A	33
3.2.2	Questão B	38

3.2.3	Questão C	40
3.3	Questões e Respostas III	43
3.3.1	Questão A	43
3.3.2	Questão B	46
3.3.3	Questão C	48
4	Conclusão	49
5	Abreviaturas	49

1 Introdução

2 Part.I do Trabalho prático

Prepare uma topologia CORE para verificar o comportamento do trace-route. Na topologia deve existir: um host (pc) cliente designado Lost, cujo router de acesso é RA1; o router RA1 está simultaneamente ligado a dois routers no core da rede RC1 e RC2; estes estão conectados a um router de acesso RA2 que, por sua vez, se liga a um host (servidor) designado Found. Apenas nas ligações (links) da rede de core, estabeleça um tempo de propagação de 15 ms.

Com base no enunciado, criamos várias estruturas de rede nomeadamente:

- um host chamado "Lost";
- dois routers de acesso "RA1" e "RA2";
- dois routers no core da rede "RC1" e "RC2";
- um host chamado "Found".

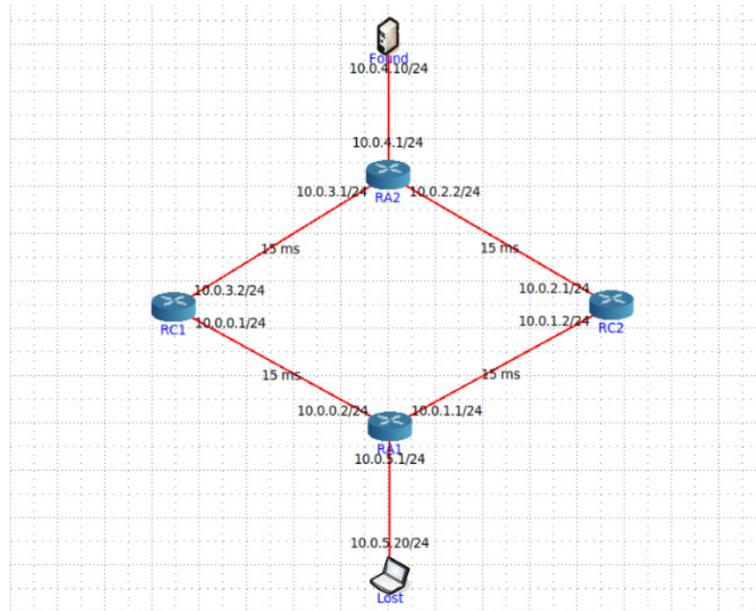


Figura 1: Topologia CORE.

2.1 Questões e Respostas I

2.1.1 Questão A

Active o Wireshark no host Lost. Numa shell de Lost execute o comando `!-I` para o endereço IP do Found. Registe e analise o tráfego ICMP enviado pelo sistema Lost e o tráfego ICMP recebido como resposta. Explique os resultados obtidos tendo em conta o princípio de funcionamento do traceroute.

Após ativar a topologia, esperou-se alguns segundos para que houvesse conectividade entre o *Lost* e o *Found*. De seguida, iniciou-se a captura de tráfego no Wireshark, a partir do nosso *host*, obtendo-se a seguinte captura:

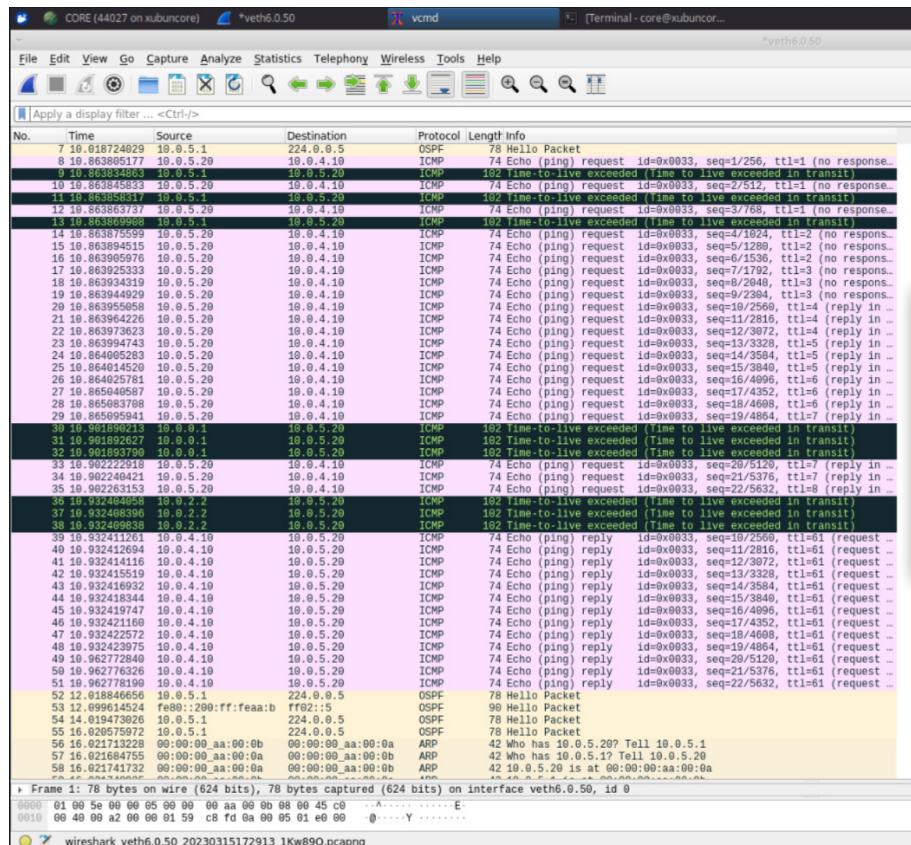
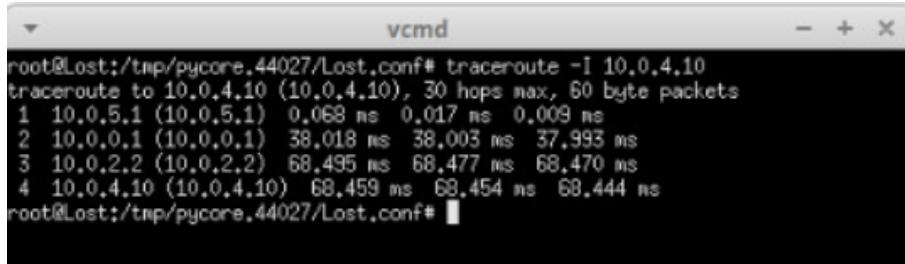


Figura 2: Captura do tráfego no Wireshark.

Após observar o tráfego no *Wireshark* efetuou-se, no terminal, o comando *traceroute -I*, tendo-se obtido o *Output* seguinte relativo ao rastreio de rotas:

A screenshot of a terminal window titled "vcmd". The window contains the command "traceroute -I 10.0.4.10" followed by its output. The output shows four routers along the path to the destination host. Router 1 is 10.0.5.1, Router 2 is 10.0.0.1, Router 3 is 10.0.2.2, and Router 4 is 10.0.4.10 (the destination). Each router entry includes its IP address, name, and the time taken for each of the three probe packets.

```
root@Lost:/tmp/pycore.44027/Lost.conf# traceroute -I 10.0.4.10
traceroute to 10.0.4.10 (10.0.4.10), 30 hops max, 60 byte packets
 1  10.0.5.1 (10.0.5.1)  0.068 ms  0.017 ms  0.009 ms
 2  10.0.0.1 (10.0.0.1)  38.018 ms  38.003 ms  37.993 ms
 3  10.0.2.2 (10.0.2.2)  68.495 ms  68.477 ms  68.470 ms
 4  10.0.4.10 (10.0.4.10)  68.459 ms  68.454 ms  68.444 ms
root@Lost:/tmp/pycore.44027/Lost.conf#
```

Figura 3: Captura do *Output* do comando *-traceroute -I*

O *traceroute* é uma ferramenta que permite descobrir a rota entre dois *hosts* numa rede local ou remota, recorrendo aos protocolos IP e ICMP e do campo TTL (*time to live*). O TTL representa, o número de saltos entre máquinas que os pacotes podem demorar numa rede, antes de serem descartados. Todas as vezes que um datagrama chega a um *router*, é decrementado o valor do seu TTL permitindo que um determinado pacote fique em circulação infinitamente (caso haja alguma falha durante o roteamento). Desta forma, sempre que um *router* recebe um datagrama com TTL igual a 0, ele já não encaminha esse pacote, descartando-o. É enviada uma mensagem que contém o endereço IP do *router*, de volta ao *host* que originou o pacote, permitindo assim identificar cada *router* por onde o pacote vai passando.

Começa-se por enviar vários datagramas (mensagens ICMP) com vários valores crescentes de TTL ao *host* de destino. Neste caso, em vez de se enviar um pacote de cada vez com TTLs crescentes, esperando por uma resposta entre cada pacote enviado, são enviados vários pacotes. Assim, é com base na resposta ICMP que se comprehende quantos TTLs são necessários para descobrir a rota dos datagramas entre os dois *hosts*. Ao chegar ao RA1, o TTL do pacote (TTL = 1) é decrementado, ficando com o valor zero, assim a mensagem ICMP “Tempo Excedido” é enviada de volta à origem, identificando-se o RA1. Depois, continuam-se a enviar mais mensagens de echo, mas surge uma nova mensagem ICMP “Tempo Excedido”, desta vez com o TTL igual a 2. Esta mensagem passou pelo RA1 (que decrementou o TTL para 1), chegando ao RC1 (que também decrementou o TTL). Então, fica-se com TTL = 0, descobrindo-se o RC1. Este processo continua até que um datagrama chegue ao *host* de destino, ou seja, até que na mensagem ICMP “Tempo Excedido” tenha um TTL = 4, já que, como irá ser referido mais à frente, na própria rede, o número de saltos mínimo entre os dois hosts é 4.

Olhando para o *output* presente na figura 3, verifica-se que o número de saltos feitos pelo *traceroute* é de 4, como se previa pela configuração da própria topologia. Também conseguimos perceber a rota realizada: 10.0.4.10 (*host Lost*) » 10.0.5.1 » 10.0.0.1 » 10.0.2.2 » 10.0.5.20 (*host Found*).

2.1.2 Questão B

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

Há 2 rotas possíveis para alcançar o servidor *Found*:

- Rota 1: *Lost* → RA1 → RC1 → RA2 → *Found*
- Rota 2: *Lost* → RA1 → RC2 → RA2 → *Found*

Ou seja, para a Rota 1:

- no salto para o router RA1, decrementa 1
 $TTL = 4 - 1 = 3$
- no salto para o router RC1, decrementa outra vez 1
 $TTL = 3 - 1 = 2$
- no salto para o router RA2, decrementa outra vez 1
 $TTL = 2 - 1 = 1$
- no último salto, alcança o servidor *Found*, que não decrementa o TTL.

Para a Rota 2:

- no salto para o router RA1, decrementa 1
 $TTL = 4 - 1 = 3$
- no salto para o router RC2, decrementa outra vez 1
 $TTL = 3 - 1 = 2$
- no salto para o router RA2, decrementa outra vez 1
 $TTL = 2 - 1 = 1$
- no último salto, alcança o servidor *Found*, que não decrementa o TTL.

Desta forma, em ambos os casos são necessários 4 saltos para chegar ao destino pretendido.

O valor inicial mínimo do campo TTL deve ser de 4 saltos entre os dois *hosts*. Considerando-se um TTL igual a 4, este é decrementado no RA1, depois no RC1, depois no RC2 e finalmente no RA2, sendo o seu valor final igual a 0 e chegando-se ao "*Found*". A figura 3 comprova o pretendido.

Para além disso, como observamos através do tráfego visualizado no *Wireshark* (figura 2), o sistema *Lost* só começa a obter resposta do sistema *Found* quando o TTL passa a ter o valor de 4. Até atingir esse valor, não é encontrada nenhuma resposta.

2.1.3 Questão 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.

Podemos obter o valor médio do RTT, fazendo a média dos valores obtidos na figura 3. Note-se que se pretende obter o tempo no acesso ao servidor, utilizando-se os valores até ao salto 4:

$$RTT = (68.459 + 68.454 + 68.444) : 3 = 68.452$$

2.1.4 Questão 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 RTT é frequentemente usado como uma aproximação do atraso, no entanto, como é uma soma dos atrasos de ida e volta, o atraso unidirecional real pode não ser estimado com precisão a partir deste.

Ao calcular o *One-Way Delay* e dividindo o RTT por 2, pressupõe-se que os caminhos de ida e volta são os mesmos em termos de congestionamento, número de saltos ou qualidade de serviço, o que normalmente não acontece, pois o atraso num sentido pode ser diferente do atraso no sentido oposto.

Assim, dividir o RTT por dois não daria um resultado preciso para o valor médio do atraso num sentido, seria apenas uma aproximação, pois esse depende da natureza da distribuição do atraso em ambas as direções: quanto mais simétricos são os atrasos nas duas direções, maior é a precisão. Ou seja, o que torna difícil o cálculo desta métrica é também o facto de os atrasos serem assimétricos. Para evitar esta má precisão, a medição pode ser feita através de um método direto que usa relógios sincronizados ou através de uma estimativa, seguindo uma fórmula específica de cálculo.

2.2 Questões e respostas II

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

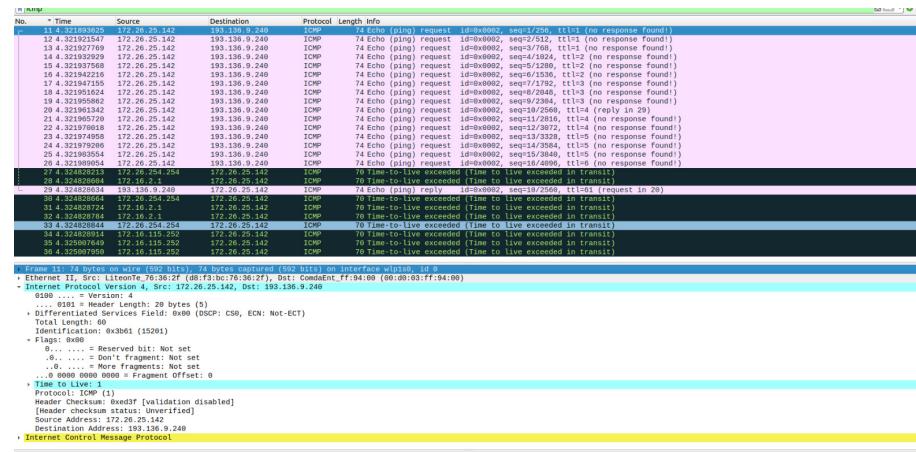


Figura 4: Captura do tráfego e detalhes da primeira mensagem ICMP capturada.

2.2.1 Questão A

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

O endereço IP, com base na figura acima é igual a 172.26.25.142.

2.2.2 Questão B

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

O valor do campo *protocol*, como se verifica na figura anterior, é ICMP (1). Este dado permite informar que o protocolo encapsulado é o ICMP. Desta forma, o número seguinte é apenas um identificador padrão utilizado para mensagens ICMP.

2.2.3 Questão C

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

Com base na figura acima, temos que o cabeçalho IPv4 tem 20 bytes (*Header Length*). O campo de dados do diagrama tem 60 bytes (*Total Length*). Desta forma, o tamanho do *payload* é calculado subtraindo o valor do cabeçalho ao do campo de dados, sendo neste caso, $60-20=40$.

2.2.4 Questão D

O datagrama IP foi fragmentado? Justifique.

O diagrama IP não foi fragmentado, pois tal como verificamos na figura 4 *Fragment offset* = 0 e *flags* = 0.

2.2.5 Questão E

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

Figura 5: Captura de tráfego nas condições referidas.

Na respetiva figura, verifica-se que entre cada conjunto de pacotes, há uma mudança no TTL, sendo essa cada vez maior. Para além disso, conseguimos observar que o valor de TTL vai mudando a cada 3 pacotes.

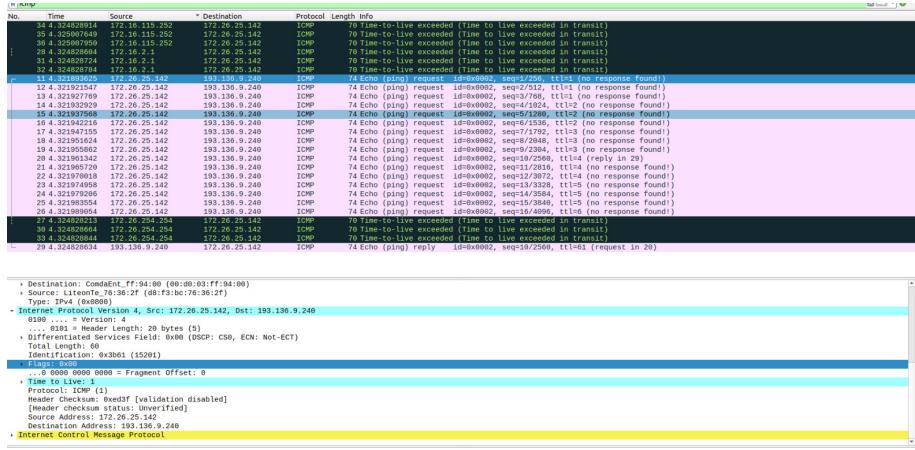


Figura 6: Captura da informação do pacote 1.

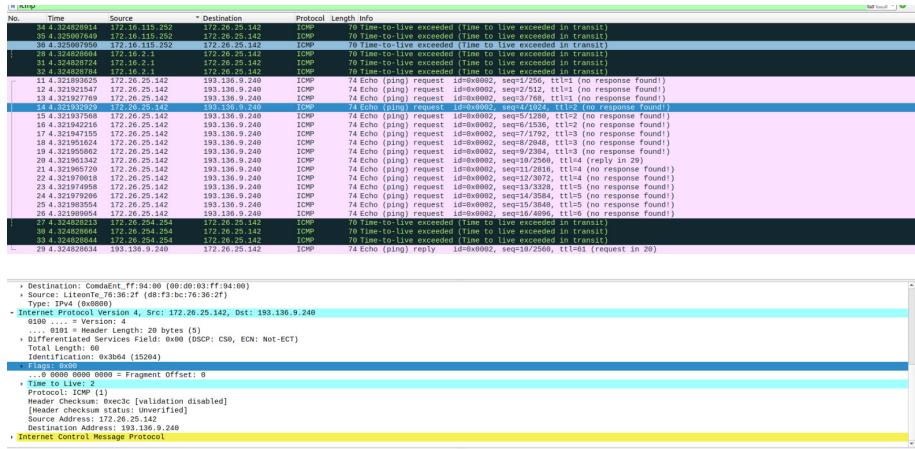


Figura 7: Captura da informação do pacote 2.

Com base nos dois pacotes acima, comprova-se que de um pacote para o outro, há uma mudança nos valores de TTL, no *identification* e no *header checksum* sendo esses os únicos campos do cabeçalho que variam de pacote para pacote.

2.2.6 Questão F

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

Tal como concluímos na pergunta anterior, são enviados pacotes com o mesmo TTL de 3 em 3. Conseguimos também perceber que o identificador de IP é incrementado em cada pacote.

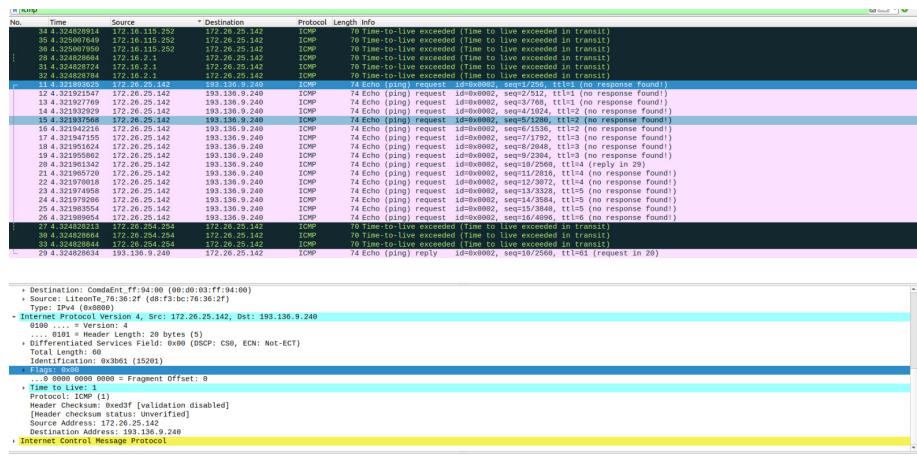


Figura 8: Captura da informação do pacote 1.

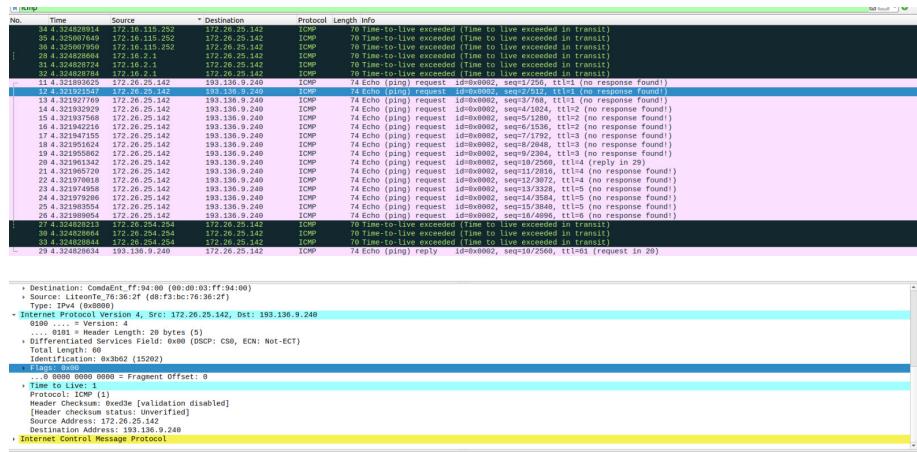


Figura 9: Captura da informação do pacote 1.

Para o efeito, selecionamos dois pacotes consecutivos, onde conseguimos notar o incremento. Olhando para os valores de Identificação do datagrama IP, nos pacotes acima, sendo estes consecutivos, o valor incrementa para um.

Para além disso, os valores, neste caso em que não há fragmentação, têm de ter valores diferentes de identificação IP. Caso isso não acontecesse, significava que os pacotes com o mesmo valor de identificação IP seriam fragmentos de um mesmo datagrama IP.

2.2.7 Questão G

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

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?

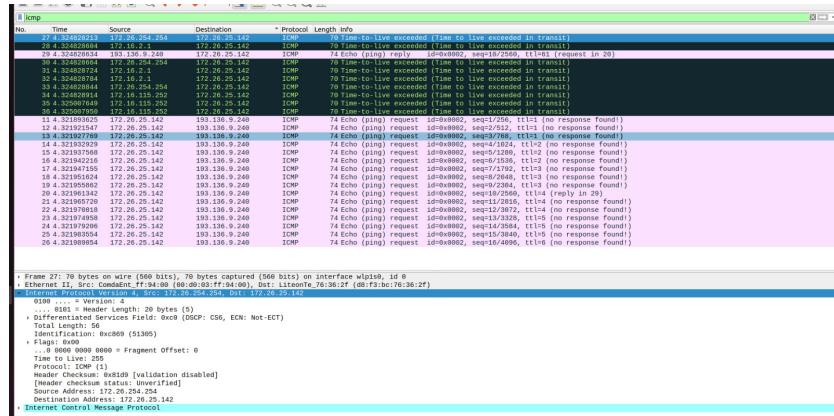


Figura 10: Captura do tráfego por organização de Destino.

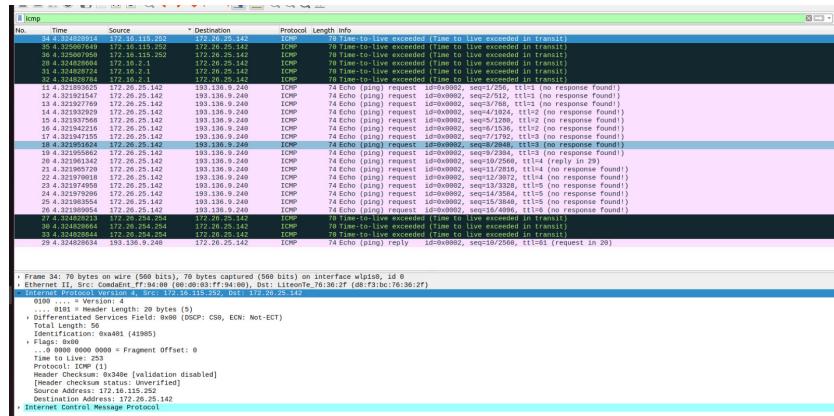


Figura 11: Captura do tráfego por organização de Source.

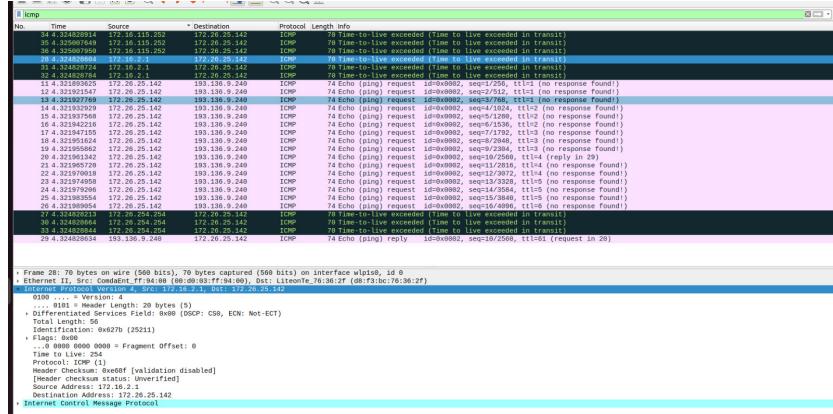


Figura 12: Captura do tráfego por organização de *Source*.

O valor do campo TTL varia. Para uma mesma *Source*, verifica-se que os primeiros três datagramas têm o campo TTL a 253, os seguintes três datagramas têm o mesmo campo a 254. Isto acontece, porque são enviados os datagramas de três em três com um valor de TTL, e de seguida outros com o valor do TTL + 1 e assim sucessivamente.

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

O valor TTL relativamente alto na mensagem de resposta é simplesmente uma precaução para garantir que a mensagem possa retornar à origem. Se o valor TTL fosse muito baixo, a mensagem de resposta poderia não conseguir chegar à origem.

2.2.8 Questão H

Sabendo que o ICMP é um protocolo pertencente ao nível de rede, discute-se a informação contida no cabeçalho ICMP poderia ser incluída no cabeçalho IPv4? Quais seriam as vantagens/desvantagens resultantes dessa hipotética inclusão?

A informação contida no cabeçalho do ICMP poderia ser incluída no cabeçalho IPv4 como uma extensão. Essa abordagem permitiria que as informações de controle e de erro contidas no ICMP fossem transportadas juntamente com o pacote IPv4, o que poderia oferecer algumas vantagens, mas também algumas desvantagens. Assim, algumas vantagens seriam:

- a redução da sobrecarga de processamento e encaminhamento em dispositivos intermediários, como roteadores, já que o processamento do cabeçalho ICMP pode ser realizado junto com o processamento do cabeçalho IPv4;
- a redução do tempo de processamento de pacotes, pois o processamento do cabeçalho ICMP seria realizado de forma mais rápida em dispositivos intermediários ou ainda a possibilidade de detetar e reportar erros de forma mais eficiente, pois as informações de erro e controle poderiam ser transportadas com o pacote IP.

Por outro lado, algumas desvantagens seriam:

- o aumento do tamanho do cabeçalho IPv4, o que pode levar à fragmentação de pacotes em redes com MTU menor;
- a possível incompatibilidade com dispositivos que não suportam a extensão ICMP, o que pode resultar em problemas de compatibilidade entre redes ou ainda o aumento do risco de ataques de negação de serviço baseados em pacotes ICMP, já que os atacantes podem usar pacotes ICMP com extensões maliciosas para sobrecarregar os dispositivos intermediários.

2.3 Questões e respostas III

2.3.1 Questão A

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

```
vicente@vicente-linux:~$ ping -s 3601 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 3601(3629) bytes of data.
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=1 ttl=61 time=3.67 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=2 ttl=61 time=4.54 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=3 ttl=61 time=11.0 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=4 ttl=61 time=4.45 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=5 ttl=61 time=8.75 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=6 ttl=61 time=9.01 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=7 ttl=61 time=8.11 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=8 ttl=61 time=5.22 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=9 ttl=61 time=7.69 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=10 ttl=61 time=4.36 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=11 ttl=61 time=4.60 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=12 ttl=61 time=4.39 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=13 ttl=61 time=11.2 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=14 ttl=61 time=12.0 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=15 ttl=61 time=3.44 ms
3609 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=16 ttl=61 time=4.63 ms
```

Figura 13: Comando de *-traceroute* com especificação 3601.

Analizando a informação relativa à primeira mensagem *ICMP Echo (ping) Request*, vemos que *Frame = 11*. Desta forma, há necessidade de fragmentação já que:

$$\text{Length(packet)} > \text{MTU} \equiv 3609 > 1480.$$

Desta forma, há fragmentação, pois o valor da Unidade de Transmissão Máxima é maior que o valor do tamanho do pacote.

Para além disso, o *Fragment offset* é 2960 (diferente de zero) e observa-se que o pacote foi fragmentado em vários datagramas IP.

Assim, foi necessária a fragmentação, pois todos os *bytes* não cabem dentro da mesma *Frame*. Assim sendo, foram necessárias três *Frames* como verificamos na figura 14.

```

    ▾ Flags: 0x01
      0... .... = Reserved bit: Not set
      .0... .... = Don't fragment: Not set
      ..0.... = More fragments: Not set
      ...0 1011 1001 0000 = Fragment Offset: 2960
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0xe460 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.25.142
  Destination Address: 193.136.9.240
  ▾ [3 IPv4 Fragments (3609 bytes): #9(1480), #10(1480), #11(649)]
    [Frame: 9, payload: 0-1479 (1480 bytes)]
    [Frame: 10, payload: 1480-2959 (1480 bytes)]
    [Frame: 11, payload: 2960-3608 (649 bytes)]
    [Fragment count: 3]
    [Reassembled IPv4 length: 3609]
    [Reassembled IPv4 data: 0800d0d2000600019616246400000000c62e0b0000000000101112131415161718191a1b...]

```

Figura 14: Captura dos detalhes do campo de fragmento.

2.3.2 Questão 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?

```

  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▾ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x016d (365)
  ▾ Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0... .... = Don't fragment: Not set
    ..1.... = More fragments: Set
    ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 64
  Protocol: ICMP (1)
  Header Checksum: 0xc293 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.25.142
  Destination Address: 193.136.9.240
  [Reassembled IPv4 in frame: 11]
  ▾ Data (1480 bytes)

```

Figura 15: Detalhes do primeiro Fragmento.

Olhando para os valores das *Flags*, conseguimos concluir que o pacote possui diversos fragmentos, pois possui o campo *More Fragments* igual a 1.

Além disso, o *Fragment offset* está a 0, o que nos indica que esse é o primeiro fragmento.

Para obter o tamanho deste datagrama IP é necessário subtrair à *Total Length* a *Header Length*, ou seja, $1500 - 20 = 1480$ bytes.

2.3.3 Questão 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?

```
0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 1500
Identification: 0x016d (365)
Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0... .... = Don't fragment: Not set
    ..1.... = More fragments: Set
...0 0101 1100 1000 = Fragment Offset: 1480
Time to Live: 64
Protocol: ICMP (1)
Header Checksum: 0xc1da [validation disabled]
[Header checksum status: Unverified]
Source Address: 172.26.25.142
Destination Address: 193.136.9.240
[Reassembled IPv4 in frame: 11]
Data (1480 bytes)
```

Figura 16: Detalhes do segundo Fragmento.

Podemos afirmar que se trata do segundo fragmento (e não do primeiro), pois o *Fragment offset* é diferente de 0, sendo igual a 1480. Podemos também concluir que existem mais fragmentos, pois o valor do campo *More Fragments* é igual a 1.

2.3.4 Questão D

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

Teoricamente podemos estimar o número de fragmentos fazendo-se a divisão do MTU - *Maximum Transmission Unit* pelo valor do tamanho do pacote:

$$3609 \div 1480 = 2,4385$$

Como o valor ultrapassa 2, arredondamos o número de fragmentos necessários a 3, sendo que o último não chegaria aos 1480 bytes.

Com base do datagrama obtido no Wireshark (e na figura 14), verificamos que o número de fragmentos é, de facto, três. Esse possui as *Frames* 23,24 e 25, sendo transportados 1480 bytes nos dois primeiros fragmentos e 649 bytes no último fragmento do datagrama.

2.3.5 Questão E

Como se deteta o último fragmento correspondente ao datagrama original? Estabeleça um filtro no Wireshark que permita listar o último fragmento do primeiro datagrama IP segmentado.

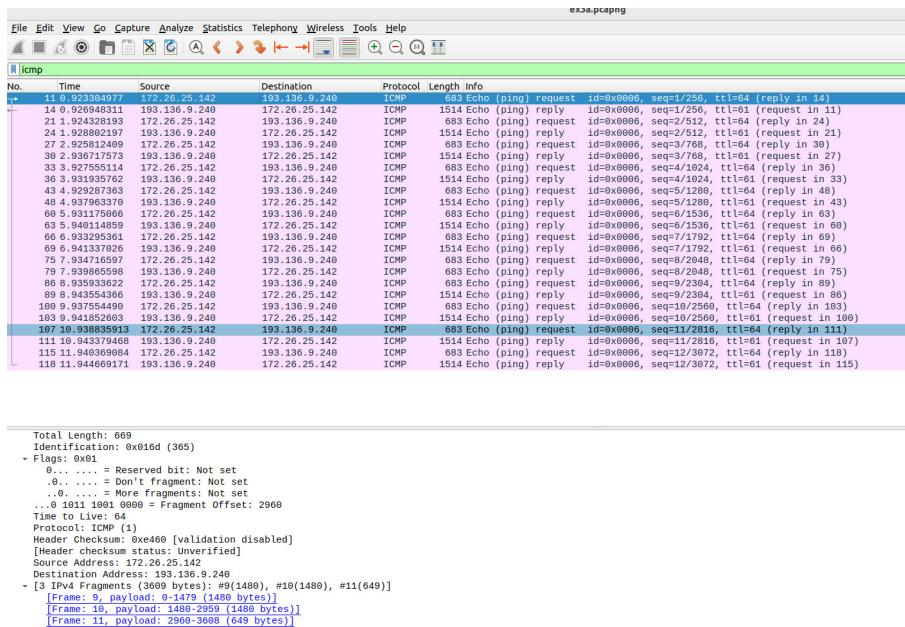


Figura 17: Cáptura de último fragmento

Com base na figura acima, podemos afirmar que se trata do último fragmento, pois o *Fragment Offset* é diferente de 0, sendo igual a 2960 e o *More Fragment* é igual a 0.

Assim, com base no filtro "ICMP" detetamos o último fragmento correspondente ao datagrama original.

2.3.6 Questão F

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

O equipamento onde o datagrama IP original é reconstruído é o destino final (*Lost*). Sendo que a reconstrução não pode ocorrer noutro equipamento, pois os fragmentos só são unidos no final, isto é, ao chegar ao destino.

2.3.7 Questão G

Indique, resumindo, os campos que mudam no cabeçalho IP entre os diferentes fragmentos, e explique a forma como essa informação permite reconstruir o datagrama original.

Os campos do cabeçalho IP que mudam entre os diversos fragmentos são os seguintes: *total length*, *flags*, *fragment offset* e o *checksum*.

No nosso caso, entre os dois primeiros pacotes e o último pacote, há uma mudança no *total length* sendo os dois primeiros iguais a 1480 bytes e o último igual a 649. Nas *Flags* (os dois primeiros fragmentos têm o bit *More fragments* a 1 e o último está a 0). O valor do *checksum* varia em cada fragmento. Finalmente, o *Fragment offset* é igual a 0 para o primeiro fragmento e diferente de 0 nos outros dois.

O fragmento identifica a *Frame*, O *More Fragments* indica-nos se existem mais fragmentos, o *Fragment Offset* identifica a ordem dos fragmentos.

O *fragment offset* e as *flags* são úteis para a reconstrução do datagrama original já que identificam a ordem de cada fragmento. Desta forma, olhando para estes campos, podemos reconstruir o datagrama original:

Ordem dos fragmentos	Fragmento (Frame)	More Fragments	Fragment Offset
1	9	1	0
2	10	1	1480
3	11	0	2960

Tabela 1: Interpretação e Reconstrução do Datagrama.

2.3.8 Questão H

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

A identificação de pacotes ICMP em fragmentos de datagramas IP é baseada no formato do cabeçalho IP. O campo *Protocol* no cabeçalho IP indica qual protocolo de camada superior que está sendo transportado no datagrama, assim se o seu valor for igual a 1, isso indica que o datagrama contém um pacote ICMP.

No entanto, quando um datagrama IP é fragmentado, o cabeçalho IP é duplicado em cada fragmento mas apenas o cabeçalho do primeiro fragmento contém o valor do campo *Protocol*. Os cabeçalhos dos fragmentos subsequentes contêm o valor do campo *Identification* e o valor do campo *Fragment Offset* para ajudar no processo de reconstrução, mas não contêm o valor do campo *Protocol*. Desta forma, os fragmentos terão o campo *Protocol* definido como 1.

Assim, apenas o primeiro fragmento de cada pacote é identificado como sendo um pacote ICMP porque contém o cabeçalho completo do protocolo de camada superior, que pode ser identificado corretamente. Os restantes fragmentos não podem ser identificados corretamente apenas com base no valor do campo *Protocol*.

Isso significa que, ao receber um fragmento, o destinatário não pode determinar qual é o protocolo de camada superior sendo transportado pelo pacote. Este precisa aguardar a chegada do primeiro fragmento para identificar o protocolo de camada superior e, em seguida, usar essa informação para reconstruir o pacote original.

2.3.9 Questão 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 MTU sendo esse igual a 1480. Assim, sendo que a *Length(packet)* é igual a 3609, esse necessita de ser fragmentado.

O aumento do MTU poderia fazer com que sejam necessários menos fragmentos ou que não fosse necessário fragmentar o datagrama. Aumentar o valor do MTU pode melhorar o desempenho da rede, pois não sendo tão fragmentado menos pacotes precisam de ser enviados para transmitir a mesma quantidade de dados. No entanto, isso pode aumentar a probabilidade de fragmentação do pacote em redes com MTUs menores, o que pode desencadear atrasos e a perda de pacotes.

Por sua vez, a diminuição desse valor faria com que fossem necessários mais fragmentos. Assim, diminuir o valor do MTU pode aumentar o tráfego de rede, pois mais pacotes precisam ser enviados para transmitir a mesma quantidade de dados. No entanto, essa diminuição pode também reduzir a probabilidade de fragmentação do pacote em redes com MTUs menores, reduzindo assim a perda de pacotes.

2.3.10 Questão J

Sabendo que no comando -ping a opção -f (Windows), -M do (Linux) ou -D (Mac) ativa a flag “Don’t Fragment” (DF) no cabeçalho do IPv4, usando ping <opção DF> <opção pkt_size> SIZE marco.uminho.pt, (opção pkt_size = -l (Windows) ou -s (Linux, Mac), determine o valor máximo de SIZE sem que ocorra fragmentação do pacote? Justifique o valor obtido.

```
vicente@vicente-linux: $ ping -M do -s 3601 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 3601(3629) bytes of data.
ping: local error: message too long, mtu=1500
ping: local error: message too long, mtu=1500
ping: local error: message too long, mtu=1500
^C
--- marco.uminho.pt ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2053ms
```

Figura 18: Cáptura do comando pedido.

O valor máximo de SIZE sem que ocorra fragmentação do pacote, com base na figura a cima, é de 3629 bytes, pois adicionamos 28 bytes de cabeçalho IP (20 bytes) e ICMP (8 bytes) ao tamanho do pacote ($3601 + 28 = 3629$).

3 Part.II do Trabalho prático

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

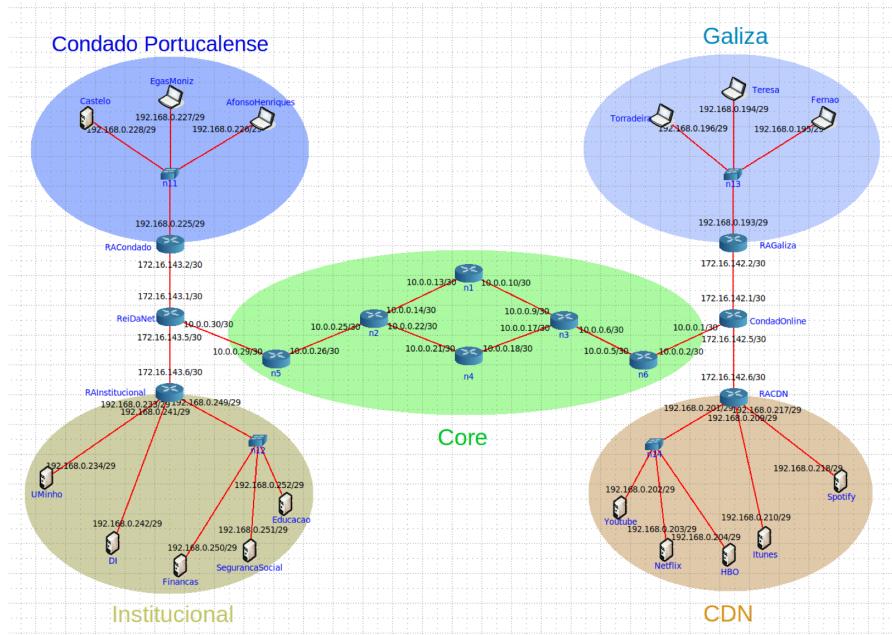


Figura 19: Topologia da rede utilizada.

3.1 Questões e Respostas I

3.1.1 Questão A

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

Ao executar o comando IP conseguimos visualizar AfonsoHenriques tem efetivamente conectividade com o servidor Financas e com os servidores da CDN.

```

Kycore,42555/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=0.329 ms
64 bytes from 192.168.0.250: icmp_seq=2 ttl=61 time=0.153 ms
64 bytes from 192.168.0.250: icmp_seq=3 ttl=61 time=0.106 ms
64 bytes from 192.168.0.250: icmp_seq=4 ttl=61 time=0.115 ms
64 bytes from 192.168.0.250: icmp_seq=5 ttl=61 time=0.156 ms
64 bytes from 192.168.0.250: icmp_seq=6 ttl=61 time=0.156 ms
64 bytes from 192.168.0.250: icmp_seq=7 ttl=61 time=0.116 ms
64 bytes from 192.168.0.250: icmp_seq=8 ttl=61 time=0.120 ms
64 bytes from 192.168.0.250: icmp_seq=9 ttl=61 time=0.099 ms
64 bytes from 192.168.0.250: icmp_seq=10 ttl=61 time=0.122 ms
64 bytes from 192.168.0.250: icmp_seq=11 ttl=61 time=0.153 ms
64 bytes from 192.168.0.250: icmp_seq=12 ttl=61 time=0.117 ms
64 bytes from 192.168.0.250: icmp_seq=13 ttl=61 time=0.256 ms
64 bytes from 192.168.0.250: icmp_seq=14 ttl=61 time=0.102 ms
64 bytes from 192.168.0.250: icmp_seq=15 ttl=61 time=0.109 ms
64 bytes from 192.168.0.250: icmp_seq=16 ttl=61 time=0.121 ms
64 bytes from 192.168.0.250: icmp_seq=17 ttl=61 time=0.107 ms
64 bytes from 192.168.0.250: icmp_seq=18 ttl=61 time=0.079 ms
64 bytes from 192.168.0.250: icmp_seq=19 ttl=61 time=0.091 ms
^C
--- 192.168.0.250 ping statistics ---
19 packets transmitted, 19 received, 0% packet loss, time 18470ms
rtt min/avg/max/mdev = 0.079/0.137/0.329/0.058 ms
root@AfonsoHenriques:/tmp/pycore,42555/AfonsoHenriques.conf# █

```

Figura 20: Ping de AfonsoHenriques para o servidor Financas

```

Kycore,42555/AfonsoHenriques.conf# ping 172.16.142.6
PING 172.16.142.6 (172.16.142.6) 56(84) bytes of data.
From 172.16.143.1 icmp_seq=1 Destination Net Unreachable
From 172.16.143.1 icmp_seq=2 Destination Net Unreachable
From 172.16.143.1 icmp_seq=3 Destination Net Unreachable
From 172.16.143.1 icmp_seq=4 Destination Net Unreachable
^C
--- 172.16.142.6 ping statistics ---
10 packets transmitted, 0 received, +4 errors, 100% packet loss, time 9216ms
root@AfonsoHenriques:/tmp/pycore,42555/AfonsoHenriques.conf# █

```

Figura 21: Ping de AfonsoHenriques para os servidores do CDN

3.1.2 Questão 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.

```

root@AfonsoHenriques:/tmp/pycore.34189/AfonsoHenriques.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         192.168.0.225  0.0.0.0       UG        0 0          0 eth0
192.168.0.224  0.0.0.0        255.255.255.248 U         0 0          0 eth0
root@AfonsoHenriques:/tmp/pycore.34189/AfonsoHenriques.conf#

```

Figura 22: Comando -netstat -rn do dispositivo AfonsoHenriques

```

root@Teresa:/tmp/pycore.34189/Teresa.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         192.168.0.193  0.0.0.0       UG        0 0          0 eth0
192.168.0.192  0.0.0.0        255.255.255.248 U         0 0          0 eth0
root@Teresa:/tmp/pycore.34189/Teresa.conf#

```

Figura 23: Comando -netstat -rn do dispositivos Teresa.

Com base na figura 22, não há problemas aparentes com as entradas para o dispositivo AfonsoHenriques.

A primeira linha indica que o endereço de destino padrão (0.0.0.0) deve ser roteado para o *gateway* 192.168.0.225 através da interface de rede eth0. Essa rota é definida com a *flag* UG, indicando que é uma rota padrão e que o *gateway* deve ser usado para encaminhar todos os pacotes de rede que não correspondem a nenhuma outra rota na tabela.

A segunda linha indica que o endereço de destino 192.168.0.224 deve ser acessado diretamente pela interface de rede eth0, sem passar por um *gateway*. A máscara de sub-rede 255.255.255.248 indica que essa rota é para uma rede local com 8 endereços IP disponíveis (192.168.0.224 a 192.168.0.231).

De modo análogo, apesar dos valores serem diferentes, não há problemas aparentes com as entradas para o dispositivo Teresa, sendo a explicação semelhante.

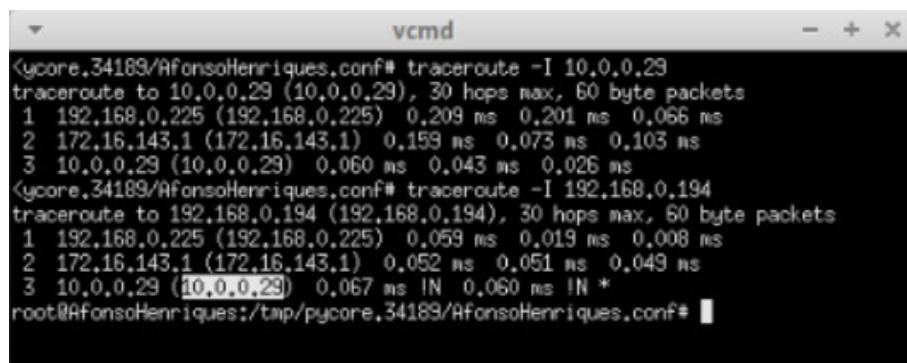
Em ambos os casos, essas rotas são importantes para garantir que os pacotes de rede sejam roteados corretamente no sistema e possam ser entregues aos seus destinos corretos.

A utilidade de cada entrada depende do contexto e da configuração específica do sistema, sendo essas maioritariamente usadas para garantir uma conectividade de rede confiável e eficiente.

3.1.3 Questão C

Utilize o Wireshark para investigar o comportamento dos routers do core da rede ($n1$ a $n6$) quando tenta estabelecer comunicação entre os hosts AfonsoHenriques e Teresa. Indique que dispositivo(s) não permite(m) o encaminhamento correto dos pacotes. Seguidamente, avalie e explique a(s) causa(s) do funcionamento incorreto do dispositivo.

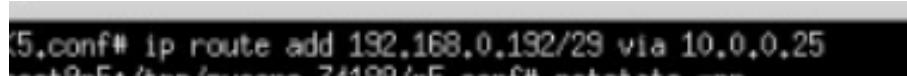
Utilize o comando `-ip route add/del` para adicionar as rotas necessárias ou remover rotas incorretas. Verifique a sintaxe completa do comando a usar com `-man ip-route` ou `man route`. Poderá também utilizar o comando traceroute para se certificar do caminho nó a nó. Considere a alínea resolvida assim que houver tráfego a chegar ao ISP CondadOnline.



```
vcmd
<ycore.34189/AfonsoHenriques.conf# traceroute -I 10.0.0.29
traceroute to 10.0.0.29 (10.0.0.29), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.209 ms  0.201 ms  0.066 ms
 2  172.16.143.1 (172.16.143.1)  0.159 ms  0.073 ms  0.103 ms
 3  10.0.0.29 (10.0.0.29)  0.060 ms  0.043 ms  0.026 ms
<ycore.34189/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)  0.059 ms  0.019 ms  0.008 ms
 2  172.16.143.1 (172.16.143.1)  0.052 ms  0.051 ms  0.049 ms
 3  10.0.0.29 (10.0.0.29)  0.067 ms !N  0.060 ms !N *
root@AfonsoHenriques:/tmp/pycore.34189/AfonsoHenriques.conf#
```

Figura 24: Comando `-traceroute`.

Com base na figura a cima, consegue-se perceber que ocorre uma falha no ip 10.0.0.29, sendo o $n5$ um dispositivo que não permite o encaminhamento correto dos pacotes. Neste caso é preciso adicionar uma rota de forma a corrigir o erro.



```
5.conf# ip route add 192.168.0.192/29 via 10.0.0.25
root@5:/tmp/pycore.34189/AfonsoHenriques.conf#
```

Figura 25: Comando `-add`.

```

root@n5:/tmp/pycore_34189/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
root@n5:/tmp/pycore_34189/n5.conf# 
```

Figura 26: Comando -netstat -rn.

```

root@fonsoferriniques:/tmp/pycore_34189/fonsoferriniques.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)  0.086 ms  0.024 ms  0.058 ms
2  172.16.143.1 (172.16.143.1)  0.061 ms  0.046 ms  0.046 ms
3  10.0.0.29 (10.0.0.29)  0.074 ms  0.056 ms  0.112 ms
4  10.0.0.25 (10.0.0.25)  0.235 ms  0.043 ms  0.040 ms
5  10.0.0.25 (10.0.0.25)  3063.198 ms  IH 3063.174 ms  3063.150 ms  IH
root@fonsoferriniques:/tmp/pycore_34189/fonsoferriniques.conf# 
```

Figura 27: Comando -traceroute.

Com base nas figuras acima, adicionamos uma nova rota. No entanto, notamos novamente que ocorre uma falha no ip 10.0.0.25, sendo o n2 o dispositivo que não permite o encaminhamento correto. Neste caso sendo que as duas rotas (do n2 para o n1 e do n2 para o n4) já existem, necessitamos de descobrir qual das duas está mal e corrigi-la.

```

root@n2:/tmp/pycore.34189/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
root@n2:/tmp/pycore.34189/n2.conf# route del -net 192.168.0.194 netmask 255.255.255.254
root@n2:/tmp/pycore.34189/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
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
root@n2:/tmp/pycore.34189/n2.conf#

```

Figura 28: Comando -netstat e Comando -del.

Com base na figura acima conseguimos perceber que a rota que desejamos remover é a de 192.168.0.194 para 10.0.0.25. pois não queremos ir do n1 para o n2.

```

root@n1:/tmp/pycore.34189/n1.conf# route del -net 192.168.0.192 netmask 255.255.255.248
root@n1:/tmp/pycore.34189/n1.conf# ip route add 192.168.0.192/29 via 10.0.0.9
root@n1:/tmp/pycore.34189/n1.conf#

```

Figura 29: Comando -del e Comando -add.

Para além disso é necessário adicionar uma nova rota, do n1 para o n3, ou seja de 10.0.0.13 para 10.0.0.9. usando o comando -add como está na figura acima.

```
root@AfonsoHenriques:/tmp/pycore_34189/AfonsoHenriques.conf# traceroute -l 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)  0.067 ms  0.009 ms  0.008 ms
 2  172.16.143.1 (172.16.143.1)  0.020 ms  0.012 ms  0.018 ms
 3  10.0.0.29 (10.0.0.29)  0.025 ms  0.016 ms  0.021 ms
 4  10.0.0.25 (10.0.0.25)  0.044 ms  0.020 ms  0.020 ms
 5  10.0.0.13 (10.0.0.13)  0.053 ms  0.029 ms  0.024 ms
 6  10.0.0.17 (10.0.0.17)  0.132 ms  0.101 ms  0.035 ms
 7  10.0.0.5 (10.0.0.5)  0.107 ms  0.031 ms  0.036 ms
 8  10.0.0.1 (10.0.0.1)  0.101 ms  0.035 ms  0.040 ms
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

Figura 30: Comando -traceroute.

Uma vez feitas essas alterações conseguimos fazer com que o tráfego chega ao ISP CondadOnline, pois chega-se ao ip 10.0.0.1 como podemos ver com base no comando -traceroute da figura acima.

3.1.4 Questão D

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

i) Em caso afirmativo, porque é que continua a não existir conectividade entre Teresa e D.Afonso Henriques? Efetue as alterações necessárias para garantir que a conectividade é restabelecida e o confronto entre os dois é evitado.

Os "*" da figura 29 indicam-nos que o programa não recebe respostas do router.

```
root@Teresa:/tmp/pycore_34189/Teresa.conf# traceroute -I 192.168.0.226
traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.106 ms !N  0.015 ms !N *
```

Figura 31: Comando -traceroute.

Desta forma, a Teresa recebe os pacotes do AfonsoHenriques, no entanto, não consegue encaminhar a resposta de que os recebeu para o AfonsoHenriques.

Assim, com base na figura 30 percebemos que ocorre uma falha no IP 192.168.0.193 e portanto, temos de adicionar a rota do RAGaliza para o CondadOnline como segue na figura abaixo.

```
root@RAGaliza:/tmp/pycore_34189/RAGaliza.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         172.16.142.1  255.255.255.252 UG        0 0          0 eth0
0.0.0.4         172.16.142.1  255.255.255.252 UG        0 0          0 eth0
0.0.0.8         172.16.142.1  255.255.255.252 UG        0 0          0 eth0
0.0.0.12        172.16.142.1  255.255.255.252 UG        0 0          0 eth0
0.0.0.16        172.16.142.1  255.255.255.252 UG        0 0          0 eth0
0.0.0.20        172.16.142.1  255.255.255.252 UG        0 0          0 eth0
0.0.0.24        172.16.142.1  255.255.255.252 UG        0 0          0 eth0
0.0.0.28        172.16.142.1  255.255.255.252 UG        0 0          0 eth0
72.0.0.0         172.16.142.1  255.0.0.0      UG        0 0          0 eth0
72.16.142.0     0.0.0.0       255.255.255.252 U          0 0          0 eth0
72.16.142.4     172.16.142.1  255.255.255.252 UG        0 0          0 eth0
72.16.143.0     172.16.142.1  255.255.255.252 UG        0 0          0 eth0
72.16.143.4     172.16.142.1  255.255.255.252 UG        0 0          0 eth0
32.168.0.192    0.0.0.0       255.255.255.248 U          0 0          0 eth1
32.168.0.200    172.16.142.1  255.255.255.248 UG        0 0          0 eth0
32.168.0.208    172.16.142.1  255.255.255.248 UG        0 0          0 eth0
32.168.0.216    172.16.142.1  255.255.255.248 UG        0 0          0 eth0
32.168.0.224    172.16.142.1  255.255.255.248 UG        0 0          0 eth0
32.168.0.232    172.16.142.1  255.255.255.248 UG        0 0          0 eth0
32.168.0.240    172.16.142.1  255.255.255.248 UG        0 0          0 eth0
32.168.0.248    172.16.142.1  255.255.255.248 UG        0 0          0 eth0
root@RAGaliza:/tmp/pycore_34189/RAGaliza.conf#
```

Figura 32: Comando -netstat -rn.

```
root@RAGaliza:/tmp/pycore.34189/RAGaliza.conf# ip route add 192.168.0.224/29 via 172.16.142.1
root@RAGaliza:/tmp/pycore.34189/RAGaliza.conf#
```

Figura 33: Comando -add.

Depois de adicionar a rota verificamos com o comando -traceroute que já se consegue encaminhar a mensagem até ao AfonsoHenriques.

```
root@Teresa:/tmp/pycore.34189/Teresa.conf# traceroute -I 192.168.0.226
traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.065 ms  0.020 ms  0.010 ms
 2  172.16.142.1 (172.16.142.1)  0.024 ms  0.023 ms  0.015 ms
 3  10.0.0.2 (10.0.0.2)  0.057 ms  0.021 ms  0.026 ms
 4  10.0.0.6 (10.0.0.6)  0.059 ms  0.025 ms  0.031 ms
 5  10.0.0.18 (10.0.0.18)  0.065 ms  0.036 ms  0.028 ms
 6  10.0.0.14 (10.0.0.14)  0.073 ms  0.072 ms  0.038 ms
 7  10.0.0.26 (10.0.0.26)  0.066 ms  0.043 ms  0.043 ms
 8  10.0.0.30 (10.0.0.30)  0.050 ms  0.040 ms  0.047 ms
 9  172.16.143.2 (172.16.143.2)  0.058 ms  0.049 ms  0.048 ms
10  192.168.0.226 (192.168.0.226)  0.074 ms  0.054 ms  0.052 ms
root@Teresa:/tmp/pycore.34189/Teresa.conf#
```

Figura 34: Comando -traceroute.

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.

```
root@AfonsoHenriques:/tmp/pycore.34189/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)  0.099 ms  0.009 ms  0.007 ms
 2  172.16.143.1 (172.16.143.1)  0.037 ms  0.012 ms  0.011 ms
 3  10.0.0.29 (10.0.0.29)  0.038 ms  0.015 ms  0.035 ms
 4  10.0.0.25 (10.0.0.25)  0.070 ms  0.032 ms  0.022 ms
 5  10.0.0.13 (10.0.0.13)  0.087 ms  0.028 ms  0.034 ms
 6  10.0.0.17 (10.0.0.17)  0.094 ms  0.123 ms  0.030 ms
 7  10.0.0.5 (10.0.0.5)  0.100 ms  0.068 ms  0.059 ms
 8  10.0.0.1 (10.0.0.1)  0.098 ms  0.045 ms  0.032 ms
 9  172.16.142.2 (172.16.142.2)  0.067 ms  0.048 ms  0.045 ms
10  192.168.0.194 (192.168.0.194)  0.097 ms  0.052 ms  0.040 ms
root@AfonsoHenriques:/tmp/pycore.34189/AfonsoHenriques.conf#
```

Figura 35: Comando -traceroute do AfonsoHenriques.

```

root@Teresa:/tmp/pycore.34189/Teresa.conf# traceroute -I 192.168.0.226
traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.065 ms  0.020 ms  0.010 ms
 2  172.16.142.1 (172.16.142.1)  0.024 ms  0.023 ms  0.015 ms
 3  10.0.0.2 (10.0.0.2)  0.057 ms  0.021 ms  0.026 ms
 4  10.0.0.6 (10.0.0.6)  0.059 ms  0.025 ms  0.031 ms
 5  10.0.0.18 (10.0.0.18)  0.065 ms  0.036 ms  0.028 ms
 6  10.0.0.14 (10.0.0.14)  0.073 ms  0.072 ms  0.038 ms
 7  10.0.0.26 (10.0.0.26)  0.066 ms  0.043 ms  0.043 ms
 8  10.0.0.30 (10.0.0.30)  0.050 ms  0.040 ms  0.047 ms
 9  172.16.143.2 (172.16.143.2)  0.058 ms  0.049 ms  0.048 ms
10  192.168.0.226 (192.168.0.226)  0.074 ms  0.054 ms  0.052 ms
root@Teresa:/tmp/pycore.34189/Teresa.conf#

```

Figura 36: Comando -traceroute da Teresa.

As rotas dos pacotes são as mesmas mas em sentido inverso, ou seja, a entrada num sentido é a saída do outro sentido e vice versa.

3.1.5 Questão E

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

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

Figura 37: Entrada sugerida pelo enunciado.

Existe uma correspondência (match) nesta entrada para pacotes enviados para o polo Galiza? E para CDN? Caso seja essa a entrada utilizada para o encaminhamento, permitirá o funcionamento esperado do dispositivo? Ofereça uma explicação pela qual essa entrada é ou não utilizada.

Não existe correspondência na entrada selecionada quando são enviados os pacotes para o polo Galiza e o polo CDN. Na entrada selecionada temos que quando o destino é a Galiza o Gateway sugerido é o 10.0.0.18, o que não acontece, pois seria como "voltar para trás". Essa entrada não iria permitir o funcionamento esperado do dispositivo, pois iria tornar-se num *loop* entre o n3 e o n4, como está explícito na figura 39.

Desta forma, o Gateway usado deve ser o 10.0.0.5.

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
10.0.0.0	10.0.0.5	255.255.255.252	UG	0	0	0	eth2
10.0.0.4	0.0.0.0	255.255.255.252	U	0	0	0	eth2
10.0.0.8	0.0.0.0	255.255.255.252	U	0	0	0	eth0
10.0.0.12	10.0.0.10	255.255.255.252	UG	0	0	0	eth0
10.0.0.16	0.0.0.0	255.255.255.252	U	0	0	0	eth1
10.0.0.20	10.0.0.18	255.255.255.252	UG	0	0	0	eth1
10.0.0.24	10.0.0.18	255.255.255.252	UG	0	0	0	eth1
10.0.0.28	10.0.0.10	255.255.255.252	UG	0	0	0	eth0
172.0.0.0	10.0.0.10	255.0.0.0	UG	0	0	0	eth0
172.16.142.0	10.0.0.5	255.255.255.252	UG	0	0	0	eth2
172.16.142.4	10.0.0.5	255.255.255.252	UG	0	0	0	eth2
172.16.143.0	10.0.0.18	255.255.255.252	UG	0	0	0	eth1
172.16.143.4	10.0.0.10	255.255.255.252	UG	0	0	0	eth0
192.168.0.192	10.0.0.5	255.255.255.248	UG	0	0	0	eth2
192.168.0.192	10.0.0.18	255.255.255.240	UG	0	0	0	eth1
192.168.0.200	10.0.0.5	255.255.255.248	UG	0	0	0	eth2
192.168.0.208	10.0.0.5	255.255.255.248	UG	0	0	0	eth2
192.168.0.216	10.0.0.5	255.255.255.248	UG	0	0	0	eth2
192.168.0.224	10.0.0.18	255.255.255.248	UG	0	0	0	eth1
192.168.0.232	10.0.0.10	255.255.255.248	UG	0	0	0	eth0
192.168.0.240	10.0.0.10	255.255.255.248	UG	0	0	0	eth0
192.168.0.248	10.0.0.10	255.255.255.248	UG	0	0	0	eth0

Figura 38: Tabela de encaminhamento de n3.

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 39: Tabela de encaminhamento de n4.

3.1.6 Questão F

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

Para determinar se uns endereços são endereços privados, devemos verificar se situam-se dentro das faixas de endereços IP privada seguintes :

- 0.0.0.0 a 10.255.255.255 (Classe A privada),
- 172.16.0.0 a 172.31.255.255 (Classe B privada),
- 192.168.0.0 a 192.168.255.255 (Classe C privada).

Desta forma, os endereços utilizados pelos quatro polos são endereços privados, pois situam-se todos dentro da Classe C privada: 192.168.0.0 a 192.168.255.255.

Os endereços utilizados no core da rede também são privados, pois situam-se todos dentro da Classe A privada: 0.0.0.0 a 10.255.255.255.

Os endereços IP privados são utilizados para comunicação interna entre dispositivos de redes locais.

3.1.7 Questão G

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

Os switches não têm um endereço IP atribuído. Isto acontece porque são dispositivos de camada 2 e portanto só são capazes de encaminhar pacotes com base no endereço MAC - *Media Access Control* dos dispositivos conectados a ele.

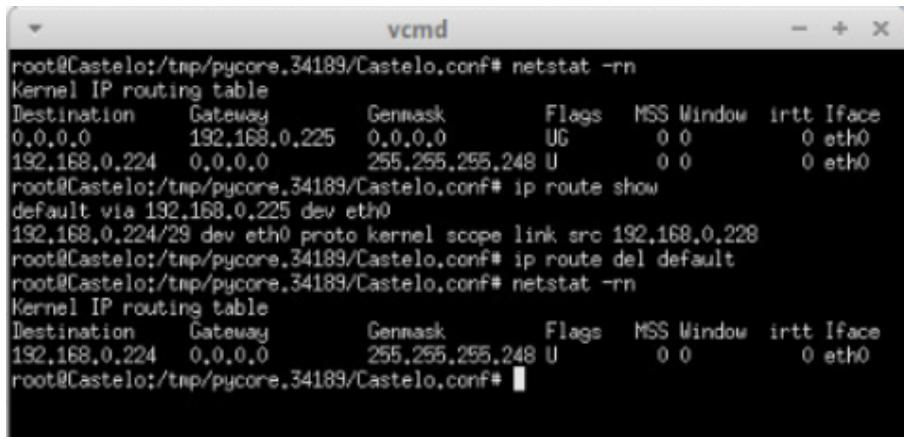
3.2 Questões e Respostas II

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

3.2.1 Questão A

Não estando satisfeito com a decoração do Castelo, opta por eliminar a sua rota default. Adicione as rotas necessárias para que o Castelo continue a ter acesso a cada um dos três polos. Mostre que a conectividade é restabelecida, assim como a tabela de encaminhamento resultante. Explique ainda a utilidade de uma rota default.

Com base no comando abaixo procurou-se a rota *default* de forma a poder ser removida previamente. Assim, uma vez encontrada removeu-se a rota como indicado na figura abaixo.



```
root@Castelo:/tmp/pycore.34189/Castelo.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         192.168.0.225  0.0.0.0        UG      0 0          0 eth0
192.168.0.224  0.0.0.0        255.255.255.248 U        0 0          0 eth0
root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route show
default via 192.168.0.225 dev eth0
192.168.0.224/29 dev eth0 proto kernel scope link src 192.168.0.228
root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route del default
root@Castelo:/tmp/pycore.34189/Castelo.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
192.168.0.224  0.0.0.0        255.255.255.248 U        0 0          0 eth0
root@Castelo:/tmp/pycore.34189/Castelo.conf#
```

Figura 40: Comandos usados de forma a encontrar a rota *default* e removê-la.

De seguida de forma a adicionar-se as rotas necessárias para que o Castelo pudesse continuar a ter acesso aos três polos, adicionou-se uma rota para o RACondado.



```
<189/Castelo.conf# ip route add 192.168.0.192/29 via 192.168.0.225
```

Figura 41: Comando -add para a Galiza.

```

root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route add 192.168.0.232/29 via>
root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route add 192.168.0.240/29 via>
root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route add 192.168.0.248/29 via>
root@Castelo:/tmp/pycore.34189/Castelo.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
192.168.0.192  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.224  0.0.0.0       255.255.255.248 U        0 0          0 eth0
192.168.0.232  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.240  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.248  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
root@Castelo:/tmp/pycore.34189/Castelo.conf# 

```

Figura 42: Comando -add para o Institucional.

```

root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route add 192.168.0.200/29 via 192.168.0.225
root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route add 192.168.0.208/29 via 192.168.0.225
root@Castelo:/tmp/pycore.34189/Castelo.conf# ip route add 192.168.0.216/29 via 192.168.0.225
root@Castelo:/tmp/pycore.34189/Castelo.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
192.168.0.192  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.200  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.208  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.216  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.224  0.0.0.0       255.255.255.248 U        0 0          0 eth0
192.168.0.232  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.240  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
192.168.0.248  192.168.0.225  255.255.255.248 UG      0 0          0 eth0
root@Castelo:/tmp/pycore.34189/Castelo.conf# 

```

Figura 43: Comando -add para o CDN.

Através das figuras abaixo podemos concluir que a conectividade entre o Castelo e os outros polos é restabelecida.

```

root@Castelo:/tmp/pycore.34189/Castelo.conf# traceroute -I 192.168.0.234
traceroute to 192.168.0.234 (192.168.0.234), 30 hops max, 60 byte packets
1 192.168.0.225 (192.168.0.225) 0.095 ms 0.010 ms 0.008 ms
2 * * *
3 * * *
4 192.168.0.234 (192.168.0.234) 0.067 ms 0.064 ms 0.060 ms
root@Castelo:/tmp/pycore.34189/Castelo.conf# traceroute -I 192.168.0.242
traceroute to 192.168.0.242 (192.168.0.242), 30 hops max, 60 byte packets
1 192.168.0.225 (192.168.0.225) 0.060 ms 0.009 ms 0.014 ms
2 * * *
3 * * *
4 192.168.0.242 (192.168.0.242) 0.083 ms 0.020 ms 0.026 ms
root@Castelo:/tmp/pycore.34189/Castelo.conf# traceroute -I 192.168.0.252
traceroute to 192.168.0.252 (192.168.0.252), 30 hops max, 60 byte packets
1 192.168.0.225 (192.168.0.225) 0.051 ms 0.009 ms 0.007 ms
2 * * *
3 * * *
4 192.168.0.252 (192.168.0.252) 0.111 ms 0.030 ms 0.018 ms

```

Figura 44: Comando -traceroute para UMinho, DI e Educação.

```

root@Castelo:/tmp/pycore_34189/Castelo.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)  0.058 ms  0.022 ms  0.013 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *

10  192.168.0.194 (192.168.0.194)  0.830 ms  0.827 ms  0.824 ms
root@Castelo:/tmp/pycore_34189/Castelo.conf# traceroute -I 192.168.0.195
traceroute to 192.168.0.195 (192.168.0.195), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.073 ms  0.024 ms  0.014 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *

10  192.168.0.195 (192.168.0.195)  0.123 ms  0.055 ms  0.049 ms
root@Castelo:/tmp/pycore_34189/Castelo.conf# traceroute -I 192.168.0.196
traceroute to 192.168.0.196 (192.168.0.196), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.040 ms  0.011 ms  0.007 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *

10  192.168.0.196 (192.168.0.196)  0.153 ms  0.054 ms  0.038 ms
root@Castelo:/tmp/pycore_34189/Castelo.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window irtt Iface
192.168.0.192    192.168.0.225  255.255.255.248 UG        0 0          0 eth0
192.168.0.224    0.0.0.0        255.255.255.248 U          0 0          0 eth0
192.168.0.232    192.168.0.225  255.255.255.248 UG        0 0          0 eth0
192.168.0.240    192.168.0.225  255.255.255.248 UG        0 0          0 eth0
192.168.0.248    192.168.0.225  255.255.255.248 UG        0 0          0 eth0

```

Figura 45: Comando -traceroute para Teresa, Torradeira e Fermão.

```

root@Castelo:/tmp/pycore_34189/Castelo.conf# traceroute -I 192.168.0.202
traceroute to 192.168.0.202 (192.168.0.202), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.062 ms  0.011 ms  0.007 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *

10  192.168.0.202 (192.168.0.202)  0.145 ms  0.082 ms  0.090 ms
root@Castelo:/tmp/pycore_34189/Castelo.conf# traceroute -I 192.168.0.210
traceroute to 192.168.0.210 (192.168.0.210), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.056 ms  0.010 ms  0.017 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *

10  192.168.0.210 (192.168.0.210)  0.182 ms  0.058 ms  0.054 ms
root@Castelo:/tmp/pycore_34189/Castelo.conf# traceroute -I 192.168.0.218
traceroute to 192.168.0.218 (192.168.0.218), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.058 ms  0.018 ms  0.007 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *

10  192.168.0.218 (192.168.0.218)  0.074 ms  0.028 ms  0.031 ms
root@Castelo:/tmp/pycore_34189/Castelo.conf# █

```

Figura 46: Comando -traceroute para Youtube, Itunes e Spotify.

Para além disso, usamos o comando -ping de forma a verificar a conectividade.

```

root@Castelo:/tmp/pycore_35629/Castelo.conf# ping 192.168.0.234
PING 192.168.0.234 (192.168.0.234) 56(84) bytes of data,
64 bytes from 192.168.0.234: icmp_seq=1 ttl=61 time=0.205 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=61 time=0.205 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=61 time=0.141 ms
64 bytes from 192.168.0.234: icmp_seq=4 ttl=61 time=0.136 ms
64 bytes from 192.168.0.234: icmp_seq=5 ttl=61 time=0.310 ms
64 bytes from 192.168.0.234: icmp_seq=6 ttl=61 time=0.205 ms
64 bytes from 192.168.0.234: icmp_seq=7 ttl=61 time=0.095 ms
64 bytes from 192.168.0.234: icmp_seq=8 ttl=61 time=0.137 ms
64 bytes from 192.168.0.234: icmp_seq=9 ttl=61 time=0.106 ms
64 bytes from 192.168.0.234: icmp_seq=10 ttl=61 time=0.108 ms
64 bytes from 192.168.0.234: icmp_seq=11 ttl=61 time=0.094 ms
64 bytes from 192.168.0.234: icmp_seq=12 ttl=61 time=0.131 ms
64 bytes from 192.168.0.234: icmp_seq=13 ttl=61 time=0.086 ms
64 bytes from 192.168.0.234: icmp_seq=14 ttl=61 time=0.079 ms
^C
--- 192.168.0.234 ping statistics ---
14 packets transmitted, 14 received, 0% packet loss, time 13313ms
rtt min/avg/max/mdev = 0.079/0.145/0.310/0.062 ms

```

Figura 47: Comando -ping do Castelo para o Institucional.

```

root@Castelo:/tmp/pycore.35629/Castelo.conf# ping 192.168.0.202
PING 192.168.0.202 (192.168.0.202) 56(84) bytes of data.
64 bytes from 192.168.0.202: icmp_seq=1 ttl=55 time=0.598 ms
64 bytes from 192.168.0.202: icmp_seq=2 ttl=55 time=0.199 ms
64 bytes from 192.168.0.202: icmp_seq=3 ttl=55 time=0.185 ms
64 bytes from 192.168.0.202: icmp_seq=4 ttl=55 time=0.190 ms
64 bytes from 192.168.0.202: icmp_seq=5 ttl=55 time=0.196 ms
64 bytes from 192.168.0.202: icmp_seq=6 ttl=55 time=0.185 ms
64 bytes from 192.168.0.202: icmp_seq=7 ttl=55 time=0.188 ms
64 bytes from 192.168.0.202: icmp_seq=8 ttl=55 time=0.184 ms
64 bytes from 192.168.0.202: icmp_seq=9 ttl=55 time=0.183 ms
^C
--- 192.168.0.202 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8202ms
rtt min/avg/max/mdev = 0.183/0.234/0.598/0.128 ms
root@Castelo:/tmp/pycore.35629/Castelo.conf# 

```

Figura 48: Comando -ping do Castelo para o CDN.

```

root@Castelo:/tmp/pycore.35629/Castelo.conf# ping 192.168.0.194
PING 192.168.0.194 (192.168.0.194) 56(84) bytes of data.
64 bytes from 192.168.0.194: icmp_seq=1 ttl=55 time=0.205 ms
64 bytes from 192.168.0.194: icmp_seq=2 ttl=55 time=0.209 ms
64 bytes from 192.168.0.194: icmp_seq=3 ttl=55 time=0.203 ms
64 bytes from 192.168.0.194: icmp_seq=4 ttl=55 time=0.235 ms
64 bytes from 192.168.0.194: icmp_seq=5 ttl=55 time=0.180 ms
64 bytes from 192.168.0.194: icmp_seq=6 ttl=55 time=0.266 ms
64 bytes from 192.168.0.194: icmp_seq=7 ttl=55 time=0.244 ms
64 bytes from 192.168.0.194: icmp_seq=8 ttl=55 time=0.257 ms
64 bytes from 192.168.0.194: icmp_seq=9 ttl=55 time=0.186 ms
64 bytes from 192.168.0.194: icmp_seq=10 ttl=55 time=0.186 ms
^C
--- 192.168.0.194 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9195ms
rtt min/avg/max/mdev = 0.180/0.217/0.266/0.029 ms
root@Castelo:/tmp/pycore.35629/Castelo.conf# poin 192.168.0.202

Command 'poin' not found, did you mean:

  command 'coin' from snap coin (5.10.0)
  command 'join' from deb coreutils (8.30-3ubuntu2)
  command 'pon' from deb ppp (2.4.7-2+4.iubuntu5.1)

See 'snap info <snapname>' for additional versions.

```

Figura 49: Comando -ping do Castelo para Galiza.

Uma rota *default* é uma rota de rede que é usada quando não há nenhuma rota mais específica disponível para encaminhar um pacote IP para um determinado destino. Desta forma, é uma rota que é usada quando o roteador não possui uma rota explícita para o destino solicitado. Assim, a sua utilidade é fundamental em redes de computadores para garantir a conectividade e o encaminhamento eficiente de pacotes IP.

3.2.2 Questão B

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

O endereço de rede IP atribuído pelo ISP é 172.16.101.128/26. Isso significa que os primeiros 26 bits do endereço são reservados para identificar a rede e os últimos 6 bits para identificar os *hosts* em cada sub-rede.

Para garantir que cada rede possa ter pelo menos 10 *hosts*, como pedido no enunciado, precisamos de uma máscara de sub-rede que forneça pelo menos 4 bits para identificar os *hosts* em cada sub-rede. Isso ocorre porque 2^4 é igual a 16, o que significa que cada sub-rede pode ter até 16 endereços IP.

Como são necessárias pelo menos 3 redes, necessitamos aumentar o número de bits usados para identificar a rede, isto é, necessitamos de pelo menos 2 bits adicionais para a identificação de rede ($2^2=4$). Logo, temos: 172.16.101.128/28.

Usando uma máscara de sub-rede de 255.255.255.240, teremos 4 bits para identificar os *hosts* em cada sub-rede, o que significa que cada sub-rede pode ter até 16 endereços IP, dos quais 14 são utilizáveis, como demonstraremos a seguir. Isso ocorre porque 2 endereços IP são reservados para o endereço de rede e o endereço de broadcast em cada sub-rede.

Agora que sabemos a máscara de sub-rede que vamos usar, podemos dividir o endereço de rede 172.16.101.128 em sub-redes. Como temos 4 bits para identificar as sub-redes, podemos criar até 16 sub-redes diferentes. Para criar as sub-redes, começamos com o primeiro endereço disponível após o endereço de rede, que é $172.16.101.128 + 1 = 172.16.101.129$. A partir daí, adicionamos 16 ao último octeto para encontrar os endereços de rede de cada sub-rede. Assim, obtemos os seguintes endereços de rede para as 3 sub-redes:

- Rede 1: 172.16.101.128;
- Rede 2: 172.16.101.144;
- Rede 3: 172.16.101.160

Cada sub-rede terá 14 endereços IP utilizáveis. Como queremos garantir que cada sub-rede possa ter pelo menos 10 *hosts*, teremos 12 endereços IP disponíveis para os dispositivos em cada sub-rede. Finalmente, para configurar os dispositivos em cada sub-rede, é necessário definir o endereço IP correto, a máscara de sub-rede e o gateway padrão. O endereço IP de um dos dispositivos em cada sub-rede pode ser definido a partir do endereço de rede da sub-rede e adicionando um número entre 1 e 14 ao

último octeto. A máscara de sub-rede será 255.255.255.240 e o *gateway* padrão será o endereço IP da sub-rede, com o último octeto definido como 1.

Para calcular o endereço de *broadcast* de uma sub-rede, precisamos mudar todos os *bits* de *host* na parte de *host* do endereço para 1s. Para encontrar o endereço de *broadcast* para as redes 1, 2 e 3, usaremos a máscara de sub-rede 255.255.255.240 e o endereço de rede correspondente a cada sub-rede:

- Endereço de rede da rede 1: 172.16.101.128
- A máscara de sub-rede em binário é : 11111111.11111111.11111111.11110000
- O último octeto do endereço de rede em binário é 10000000
- Para encontrar o endereço de broadcast, mudamos todos os *bits* de host para 1s, o que nos dá 172.16.101.143

Segue na tabela abaixo o mesmo para as redes 2 e 3.

Sub-rede	Endereço de rede	Primeiro endereço IP	Broadcast	Ultimo endereço
1	172.16.101.128	172.16.101.129	172.16.101.143	172.16.101.142
2	172.16.101.144	172.16.101.145	172.16.101.159	172.16.101.158
3	172.16.101.160	172.16.101.161	172.16.101.175	172.16.101.174

Tabela 2: Esquema de Endereçamento

Como referido anteriormente, usando a máscara de sub-rede 255.255.255.240, cada sub-rede terá um total de 16 endereços IP disponíveis. Então, a quantidade de endereços disponíveis em cada sub-rede é:

- Rede 1: 16 endereços possíveis - 2 endereços reservados = 14 endereços disponíveis;
- Rede 2: 16 endereços possíveis - 2 endereços reservados = 14 endereços disponíveis;
- Rede 3: 16 endereços possíveis - 2 endereços reservados = 14 endereços disponíveis.

3.2.3 Questão C

Ligue um novo host diretamente ao router ReiDaNet. Associe-lhe um endereço, à sua escolha, pertencente a uma sub-rede disponível das criadas na alínea anterior (garanta que a interface do router ReiDaNet utiliza o primeiro endereço da sub-rede escolhida). Verifique que tem conectividade com os diferentes polos. Existe algum host com o qual não seja possível comunicar? Porquê? Se reiniciou a simulação, repita todas as alterações efetuadas anteriormente e responda a esta questão a partir desse estado.

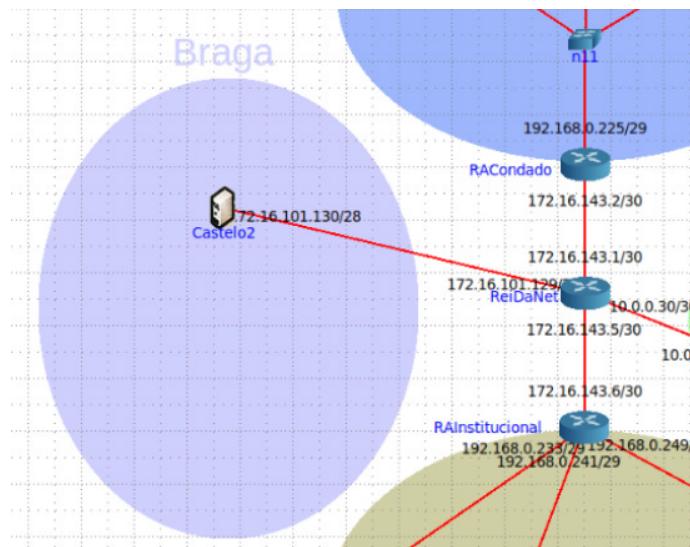


Figura 50: NovoHost-CasteloI ligado ao ReiDaNet.

De forma ao endereço do novo *host* - CateloI pertencer a uma sub-rede disponível das criadas na alínea anterior escolheu-se o endereço IP 172.16.101.130.

De forma a verificar a conectividade com os outros polos utilizou-se o comando - netstat -rn verificando se existia uma rota *default* que ligava o *host* ao *router* ReiDaNet 172.16.101.129.

```

root@Castelo2:/tmp/pycore.35629/Castelo2.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window irtt Iface
0.0.0.0        172.16.101.129  0.0.0.0        UG        0 0          0 eth0
172.16.101.128 0.0.0.0        255.255.255.240 U          0 0          0 eth0

```

Figura 51: Comando -netstat -rn.

Conseguimos identificar a rota default com base na figura acima referida, sendo essa a que possui *Destination* e *Genmask* 0.0.0.0.

```

root@Castelo2:/tmp/pycore.35629/Castelo2.conf# traceroute -I 192.168.0.234
traceroute to 192.168.0.234 (192.168.0.234), 30 hops max, 60 byte packets
1 172.16.101.129 (172.16.101.129) 0.114 ms 0.021 ms 0.012 ms
2 172.16.143.6 (172.16.143.6) 0.051 ms 0.033 ms 0.019 ms
3 192.168.0.234 (192.168.0.234) 0.051 ms 0.038 ms 0.023 ms

```

Figura 52: Exemplo de conectividade com o Institucional - host UMinho.

```

root@Castelo2:/tmp/pycore.35629/Castelo2.conf# traceroute -I 192.168.0.202
traceroute to 192.168.0.202 (192.168.0.202), 30 hops max, 60 byte packets
1 172.16.101.129 (172.16.101.129) 0.056 ms 0.014 ms 0.010 ms
2 10.0.0.29 (10.0.0.29) 0.038 ms 0.016 ms 0.024 ms
3 10.0.0.25 (10.0.0.25) 0.031 ms 0.029 ms 0.021 ms
4 10.0.0.21 (10.0.0.21) 0.046 ms 0.035 ms 0.026 ms
5 10.0.0.9 (10.0.0.9) 0.102 ms 0.043 ms 0.032 ms
6 10.0.0.5 (10.0.0.5) 0.060 ms 0.061 ms 0.048 ms
7 10.0.0.1 (10.0.0.1) 0.062 ms 0.053 ms 0.052 ms
8 172.16.142.6 (172.16.142.6) 0.094 ms 0.059 ms 0.050 ms
9 192.168.0.202 (192.168.0.202) 0.072 ms 0.063 ms 0.062 ms

```

Figura 53: Exemplo de Conectividade com o CDN - host Youtube.

```
J 192.168.0.194 (192.168.0.194) 0.072 ms 0.008 ms 0.006 ms
root@Castelo2:/tmp/pycore.35629/Castelo2.conf# traceroute -I 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
1 172.16.101.129 (172.16.101.129) 0.042 ms 0.008 ms 0.006 ms
2 10.0.0.29 (10.0.0.29) 0.022 ms 0.017 ms 0.010 ms
3 10.0.0.25 (10.0.0.25) 0.022 ms 0.011 ms 0.020 ms
4 10.0.0.13 (10.0.0.13) 0.027 ms 0.022 ms 0.015 ms
5 10.0.0.9 (10.0.0.9) 0.028 ms 0.018 ms 0.017 ms
6 10.0.0.5 (10.0.0.5) 0.039 ms 0.067 ms 0.022 ms
7 10.0.0.1 (10.0.0.1) 0.040 ms 0.023 ms 0.030 ms
8 172.16.142.2 (172.16.142.2) 0.046 ms 0.027 ms 0.033 ms
9 192.168.0.194 (192.168.0.194) 0.039 ms 0.030 ms 0.036 ms
root@Castelo2:/tmp/pycore.35629/Castelo2.conf# traceroute -I 192.168.0.226
```

Figura 54: Exemplo de Conectividade com a Galiza - host Teresa.

```
J 192.168.0.194 (192.168.0.194) 0.059 ms 0.009 ms 0.006 ms
root@Castelo2:/tmp/pycore.35629/Castelo2.conf# traceroute -I 192.168.0.226
traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
1 172.16.101.129 (172.16.101.129) 0.048 ms 0.010 ms 0.007 ms
2 172.16.143.2 (172.16.143.2) 0.042 ms 0.018 ms 0.028 ms
3 192.168.0.226 (192.168.0.226) 0.035 ms 0.035 ms 0.021 ms
root@Castelo2:/tmp/pycore.35629/Castelo2.conf# traceroute -I 192.168.0.226
```

Figura 55: Exemplo de Conectividade com o Condado Portugalense - host AfonsoHenriques.

O NovoHost tem conectividade com todos os polos e mais especificamente todos os *hosts* (como referido nos exemplos acima) com a exceção do Castelo.

```
root@Castelo2:/tmp/pycore.35629/Castelo2.conf# traceroute -I 192.168.0.228
traceroute to 192.168.0.228 (192.168.0.228), 30 hops max, 60 byte packets
1 172.16.101.129 (172.16.101.129) 0.073 ms 0.041 ms 0.015 ms
2 172.16.143.2 (172.16.143.2) 0.024 ms 0.010 ms 0.009 ms
3 ***
4 ***
5 ***
6 ***
7 ***
8 ***
9 ***
10 ***
11 ***
12 ***
13 ***
14 ***
15 ***
16 ***
17 ***
18 ***
19 ***
20 ***
21 ***
22 ***
23 ***
24 ***
25 ***
26 ***
27 ***
28 ***
29 ***
30 ***
root@Castelo2:/tmp/pycore.35629/Castelo2.conf#
```

Figura 56: Comando traceroute do NovoHost para o host Castelo.

```

root@Castelo:/tmp/pycore.35629/Castelo.conf# traceroute -I 172.16.101.130
connect: Network is unreachable
root@Castelo:/tmp/pycore.35629/Castelo.conf#

```

Figura 57: Comando traceroute do Castelo para o NovoHost.

Com base nos dois comandos das figuras 56 e 57, conseguimos concluir que o *host* Castelo recebe as mensagens do NovoHost. No entanto, sendo que o Castelo não possui rota *default* este não consegue responder ao NovoHost, não havendo conectividade.

3.3 Questões e Respostas III

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

3.3.1 Questão A

De modo a facilitar a travessia, elimine as rotas referentes a Galiza e CDN no dispositivo n6 e defina um esquema de summarização de rotas (Supernetting) que permita o uso de apenas uma rota para ambos os polos. Confirme que a conectividade é mantida.

Primeiramente, com base na figura abaixo, fez se o comando -netstat -rn do n6 de forma a ver os diversos caminhos existentes.

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
10.0.0.0	0.0.0.0	255.255.255.252	U	0	0		0 eth0
10.0.0.4	0.0.0.0	255.255.255.252	U	0	0		0 eth1
10.0.0.8	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
10.0.0.12	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
10.0.0.16	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
10.0.0.20	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
10.0.0.24	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
10.0.0.28	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
172.0.0.0	10.0.0.6	255.0.0.0	UG	0	0		0 eth1
172.16.142.0	10.0.0.1	255.255.255.252	UG	0	0		0 eth0
172.16.142.4	10.0.0.1	255.255.255.252	UG	0	0		0 eth0
172.16.143.0	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
172.16.143.4	10.0.0.6	255.255.255.252	UG	0	0		0 eth1
192.168.0.192	10.0.0.1	255.255.255.248	UG	0	0		0 eth0
192.168.0.200	10.0.0.1	255.255.255.248	UG	0	0		0 eth0
192.168.0.208	10.0.0.1	255.255.255.248	UG	0	0		0 eth0
192.168.0.216	10.0.0.1	255.255.255.248	UG	0	0		0 eth0
192.168.0.224	10.0.0.6	255.255.255.248	UG	0	0		0 eth1
192.168.0.232	10.0.0.6	255.255.255.248	UG	0	0		0 eth1
192.168.0.240	10.0.0.6	255.255.255.248	UG	0	0		0 eth1
192.168.0.248	10.0.0.6	255.255.255.248	UG	0	0		0 eth1

Figura 58: Comando -netstat -rn.

De seguida, realizou-se a remoção das rotas referentes a Galiza e ao CDN, como se pode observar na figura 59.

Para além disso, adicionou-se a rota de rede 192.168.0.192/27, pois a máscara 27 de sub-rede consegue cobrir todas as redes.

```
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.192 via 10.0.0.1
RTNETLINK answers: No such process
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.192/29 via 10.0.0.1
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.200/29 via 10.0.0.1
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.208/29 via 10.0.0.1
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.216/29 via 10.0.0.1
root@n6:/tmp/pycore.35629/n6.conf# ip route add 192.168.0.192/27 via 10.0.0.1
root@n6:/tmp/pycore.35629/n6.conf# netstat -rn
```

Figura 59: Comandos -del e -add.

Finalmente, fez-se novamente comando -netstat -rn e o -traceroute de forma a verificar a conectividade.

```
root@n6:/tmp/pycore.35629/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      255.255.255.248 UG       0 0          0 eth1
192.168.0.232    10.0.0.6      255.255.255.248 UG       0 0          0 eth1
192.168.0.240    10.0.0.6      255.255.255.248 UG       0 0          0 eth1
192.168.0.248    10.0.0.6      255.255.255.248 UG       0 0          0 eth1
root@n6:/tmp/pycore.35629/n6.conf# netstat -rn
```

Figura 60: Comando -netstat -rn.

```

pycore_35629@Rfonsinhoenriques.conf# traceroute -I 192.168.0.194
traceroute to 192.168.0.194 (192.168.0.194), 30 hops max, 60 byte packets
1 192.168.0.226 (192.168.0.226) 0.04 ms 0.017 ms 0.022 ms
2 172.16.143.1 (172.16.143.1) 0.047 ms 0.025 ms 0.040 ms
3 10.0.0.29 (10.0.0.29) 0.055 ms 0.048 ms 0.041 ms
4 10.0.0.26 (10.0.0.26) 0.068 ms 0.056 ms 0.053 ms
5 10.0.0.13 (10.0.0.13) 0.115 ms 0.064 ms 0.062 ms
6 10.0.0.17 (10.0.0.17) 0.204 ms 0.104 ms 0.056 ms
7 10.0.0.5 (10.0.0.5) 0.110 ms 0.064 ms 0.073 ms
8 10.0.0.1 (10.0.0.1) 0.100 ms 0.081 ms 0.085 ms
9 172.16.142.2 (172.16.142.2) 0.098 ms 0.099 ms 0.084 ms
10 192.168.0.194 (192.168.0.194) 0.127 ms 0.092 ms 0.093 ms
</>#Rfonsinhoenriques.conf# traceroute -I 192.168.0.218
traceroute to 192.168.0.218 (192.168.0.218), 30 hops max, 60 byte packets
1 192.168.0.225 (192.168.0.225) 0.122 ms 0.017 ms 0.013 ms
2 172.16.143.1 (172.16.143.1) 0.049 ms 0.022 ms 0.036 ms
3 10.0.0.29 (10.0.0.29) 0.062 ms 0.043 ms 0.027 ms
4 10.0.0.25 (10.0.0.25) 0.102 ms 0.039 ms 0.036 ms
5 10.0.0.21 (10.0.0.21) 0.137 ms 0.061 ms 0.059 ms
6 10.0.0.17 (10.0.0.17) 0.128 ms 0.174 ms 0.083 ms
7 10.0.0.5 (10.0.0.5) 0.198 ms 0.298 ms 0.210 ms
8 10.0.0.1 (10.0.0.1) 0.279 ms 0.176 ms 0.077 ms
9 172.16.142.2 (172.16.142.2) 0.075 ms 0.048 ms 0.046 ms
10 192.168.0.218 (192.168.0.218) 0.150 ms 0.178 ms 0.069 ms
</>#Rfonsinhoenriques.conf# traceroute -I 192.168.0.204
traceroute to 192.168.0.204 (192.168.0.204), 30 hops max, 60 byte packets
1 192.168.0.226 (192.168.0.226) 0.111 ms 0.050 ms 0.048 ms
2 172.16.143.1 (172.16.143.1) 0.068 ms 0.056 ms 0.149 ms
3 10.0.0.29 (10.0.0.29) 0.063 ms 0.070 ms 0.073 ms
4 10.0.0.25 (10.0.0.25) 0.118 ms 0.089 ms 0.181 ms
5 10.0.0.21 (10.0.0.21) 0.068 ms 0.073 ms 0.059 ms
6 10.0.0.17 (10.0.0.17) 0.077 ms 0.196 ms 0.131 ms
7 10.0.0.5 (10.0.0.5) 0.155 ms 0.136 ms 0.159 ms
8 10.0.0.1 (10.0.0.1) 0.188 ms 0.154 ms 0.081 ms
9 172.16.142.6 (172.16.142.6) 0.096 ms 0.093 ms 0.067 ms
10 192.168.0.204 (192.168.0.204) 0.114 ms 0.104 ms 0.129 ms
</>#Rfonsinhoenriques.conf# traceroute -I 192.168.0.210
traceroute to 192.168.0.210 (192.168.0.210), 30 hops max, 60 byte packets
1 192.168.0.225 (192.168.0.225) 0.052 ms 0.017 ms 0.030 ms
2 172.16.143.1 (172.16.143.1) 0.038 ms 0.028 ms 0.033 ms
3 10.0.0.29 (10.0.0.29) 0.059 ms 0.081 ms 0.040 ms
4 10.0.0.25 (10.0.0.25) 0.068 ms 0.049 ms 0.046 ms
5 10.0.0.21 (10.0.0.21) 0.069 ms 0.055 ms 0.058 ms
6 10.0.0.17 (10.0.0.17) 0.078 ms 0.106 ms 0.058 ms
7 10.0.0.1 (10.0.0.1) 0.109 ms 0.084 ms 0.075 ms
8 172.16.142.6 (172.16.142.6) 0.098 ms 0.086 ms 0.109 ms
10 192.168.0.210 (192.168.0.210) 0.137 ms 0.098 ms 0.091 ms
root@Rfonsinhoenriques:~# /tmp/pycore_35629/Rfonsinhoenriques.conf# 

```

Figura 61: Comando -traceroute para Teresa, Spotify, HBO e Itunes.

```

root@n6:/tmp/pycore_35629/n6.conf# ping 192.168.0.204
PING 192.168.0.204 (192.168.0.204) 56(84) bytes of data.
64 bytes from 192.168.0.204: icmp_seq=1 ttl=62 time=0.133 ms
64 bytes from 192.168.0.204: icmp_seq=2 ttl=62 time=0.091 ms
64 bytes from 192.168.0.204: icmp_seq=3 ttl=62 time=0.113 ms
64 bytes from 192.168.0.204: icmp_seq=4 ttl=62 time=0.050 ms
64 bytes from 192.168.0.204: icmp_seq=5 ttl=62 time=0.112 ms
64 bytes from 192.168.0.204: icmp_seq=6 ttl=62 time=0.142 ms
64 bytes from 192.168.0.204: icmp_seq=7 ttl=62 time=0.108 ms
C
--- 192.168.0.204 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6161ms
rtt min/avg/max/mdev = 0.050/0.107/0.142/0.027 ms
root@n6:/tmp/pycore_35629/n6.conf# 

```

Figura 62: Comando -ping para HBO.

3.3.2 Questão B

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

Tal como na pergunta anterior, inicialmente, utilizamos o comando -netstat -rn do n6 de forma a ver os diversos caminhos existentes.

De seguida, realizou-se a remoção das rotas referentes ao Condado Portucalence e ao Institucional, como se pode observar na figura abaixo.

Para além disso, adicionou-se a rota de rede 192.168.0.224/27, pois tal como referido anteriormente, a máscara 27 de sub-rede consegue cobrir todas as redes. Para concluir, fez-se novamente comando -netstat -rn e o -traceroute de forma a verificar a conectividade.

```
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.224/29 via 10.0.0.6
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.232/29 via 10.0.0.6
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.240/29 via 10.0.0.6
root@n6:/tmp/pycore.35629/n6.conf# ip route del 192.168.0.248/29 via 10.0.0.6
root@n6:/tmp/pycore.35629/n6.conf# ip route add 192.168.0.248/27 via 10.0.0.6
Error: Invalid prefix for given prefix length.
root@n6:/tmp/pycore.35629/n6.conf# ip route add 192.168.0.224/27 via 10.0.0.6
root@n6:/tmp/pycore.35629/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      256.255.255.224 UG     0 0          0 eth0
192.168.0.224    10.0.0.6      255.255.255.224 UG     0 0          0 eth1
root@n6:/tmp/pycore.35629/n6.conf#
```

Figura 63: Comandos -del e -add.

```

root@Teresa:/tmp/pycore_35629/Teresa.conf# traceroute -I 192.168.0.226
traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.102 ms  0.021 ms  0.032 ms
 2  172.16.142.1 (172.16.142.1)  0.053 ms  0.031 ms  0.029 ms
 3  10.0.0.2 (10.0.0.2)  0.095 ms  0.054 ms  0.052 ms
 4  10.0.0.6 (10.0.0.6)  0.175 ms  0.065 ms  0.063 ms
 5  10.0.0.18 (10.0.0.18)  0.195 ms  0.083 ms  0.072 ms
 6  10.0.0.14 (10.0.0.14)  0.153 ms  0.097 ms  0.073 ms
 7  10.0.0.26 (10.0.0.26)  0.125 ms  0.093 ms  0.063 ms
 8  10.0.0.30 (10.0.0.30)  0.113 ms  0.107 ms  0.100 ms
 9  172.16.143.2 (172.16.143.2)  0.121 ms  0.121 ms  0.120 ms
10  192.168.0.226 (192.168.0.226)  0.178 ms  0.130 ms  0.162 ms
root@Teresa:/tmp/pycore_35629/Teresa.conf# traceroute -I 192.168.0.234
traceroute to 192.168.0.234 (192.168.0.234), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.041 ms  0.008 ms  0.014 ms
 2  172.16.142.1 (172.16.142.1)  0.018 ms  0.011 ms  0.012 ms
 3  10.0.0.2 (10.0.0.2)  0.030 ms  0.016 ms  0.021 ms
 4  10.0.0.6 (10.0.0.6)  0.023 ms  0.016 ms  0.022 ms
 5  10.0.0.14 (10.0.0.14)  0.028 ms  0.026 ms  0.019 ms
 6  10.0.0.14 (10.0.0.14)  0.038 ms  0.053 ms  0.027 ms
 7  10.0.0.26 (10.0.0.26)  0.045 ms  0.036 ms  0.031 ms
 8  10.0.0.30 (10.0.0.30)  0.046 ms  0.040 ms  0.035 ms
 9  172.16.143.6 (172.16.143.6)  0.047 ms  0.037 ms  0.032 ms
10  192.168.0.234 (192.168.0.234)  0.056 ms  0.043 ms  0.035 ms
root@Teresa:/tmp/pycore_35629/Teresa.conf# traceroute -I 192.168.0.242
traceroute to 192.168.0.242 (192.168.0.242), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.035 ms  0.009 ms  0.014 ms
 2  172.16.142.1 (172.16.142.1)  0.019 ms  0.012 ms  0.018 ms
 3  10.0.0.2 (10.0.0.2)  0.024 ms  0.015 ms  0.020 ms
 4  10.0.0.6 (10.0.0.6)  0.025 ms  0.024 ms  0.019 ms
 5  10.0.0.10 (10.0.0.10)  0.029 ms  0.023 ms  0.023 ms
 6  10.0.0.14 (10.0.0.14)  0.037 ms  0.065 ms  0.032 ms
 7  10.0.0.26 (10.0.0.26)  0.038 ms  0.035 ms  0.029 ms
 8  10.0.0.30 (10.0.0.30)  0.047 ms  0.040 ms  0.039 ms
 9  172.16.143.6 (172.16.143.6)  0.045 ms  0.040 ms  0.039 ms
10  192.168.0.242 (192.168.0.242)  0.072 ms  0.048 ms  0.044 ms
root@Teresa:/tmp/pycore_35629/Teresa.conf# traceroute -I 192.168.0.252
traceroute to 192.168.0.252 (192.168.0.252), 30 hops max, 60 byte packets
 1  192.168.0.193 (192.168.0.193)  0.053 ms  0.020 ms  0.017 ms
 2  172.16.142.1 (172.16.142.1)  0.032 ms  0.023 ms  0.023 ms
 3  10.0.0.2 (10.0.0.2)  0.038 ms  0.030 ms  0.028 ms
 4  10.0.0.6 (10.0.0.6)  0.046 ms  0.032 ms  0.032 ms
 5  10.0.0.10 (10.0.0.10)  0.050 ms  0.038 ms  0.045 ms
 6  10.0.0.14 (10.0.0.14)  0.053 ms  0.205 ms  0.158 ms
 7  10.0.0.26 (10.0.0.26)  0.185 ms  0.174 ms  0.171 ms
 8  10.0.0.30 (10.0.0.30)  0.212 ms  0.170 ms  0.093 ms
 9  172.16.143.6 (172.16.143.6)  0.104 ms  0.098 ms  0.108 ms
10  192.168.0.252 (192.168.0.252)  0.179 ms  0.153 ms  0.115 ms
root@Teresa:/tmp/pycore_35629/Teresa.conf# 

```

Figura 64: Comando -traceroute para AfonsoHenriques, UMinho, DI e Educação.

```

root@n6:/tmp/pycore_35629/n6.conf# ping 192.168.0.226
PING 192.168.0.226 (192.168.0.226) 56(84) bytes of data.
64 bytes from 192.168.0.226: icmp_seq=1 ttl=58 time=0.165 ms
64 bytes from 192.168.0.226: icmp_seq=2 ttl=58 time=0.174 ms
64 bytes from 192.168.0.226: icmp_seq=3 ttl=58 time=0.173 ms
64 bytes from 192.168.0.226: icmp_seq=4 ttl=58 time=0.159 ms
64 bytes from 192.168.0.226: icmp_seq=5 ttl=58 time=0.172 ms
64 bytes from 192.168.0.226: icmp_seq=6 ttl=58 time=0.155 ms
64 bytes from 192.168.0.226: icmp_seq=7 ttl=58 time=0.168 ms
^C
--- 192.168.0.226 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6131ms
rtt min/avg/max/mdev = 0.155/0.166/0.174/0.006 ms
root@n6:/tmp/pycore_35629/n6.conf# 

```

Figura 65: Comando -ping para AfonsoHenriques.

3.3.3 Questão C

Comente os aspectos positivos e negativos do uso do Supernetting.

O *Supernetting* é uma técnica de design de rede que envolve a combinação de várias redes IP menores em uma única rede maior, para reduzir a complexidade e o tamanho das tabelas de roteamento.

Desta forma, agrupa-se várias redes menores numa única rede maior, reduzindo o número de entradas na tabela de roteamento do sistema autônomo ou roteador. Em vez de manter várias rotas para cada sub-rede individualmente, o roteador pode manter uma única rota para a rede supernetizada.

Alguns exemplos de aspectos positivos do *Supernetting* são:

- Redução do tamanho da tabela de roteamento, pois tal como referido anteriormente, com o *Supernetting*, várias sub-redes podem ser agrupadas numa única rede maior, o que pode reduzir o tamanho da tabela de roteamento. Assim, pode-se melhorar o desempenho da rede e reduzir a carga de processamento nos roteadores.
- Otimização do uso de endereços IP, pois ao agrupar várias sub-redes numa única rede maior, pode-se otimizar o uso de endereços IP, já que menos endereços IP são necessários para representar a rede agregada.

Alguns exemplos de aspectos negativos do *Supernetting* são:

- Perda de flexibilidade, pois quando várias sub-redes são agregadas numa só rede maior, a flexibilidade da rede pode ser perdida. Por exemplo, se uma sub-rede precisar ser alterada ou removida, pode ser necessário reconfigurar toda a rede agregada.
- Falha de rede, pois se uma rede agregada for interrompida, todas as sub-redes contidas nessa rede também serão interrompidas. Isso pode aumentar o risco de falhas de rede e dificultar a resolução de problemas de rede.

4 Conclusão

Este trabalho foi dividido em duas partes gerais, cada uma com um grau de dificuldade e complexidade crescente. Na primeira parte, analisamos o tráfego utilizando diversos comandos como o -traceroute e explorando conceitos como TTL, RTT e os protocolos ICMP e IP. Além disso, a utilização do Wireshark permitiu estabelecer uma relação entre a topologia, o comando executado no terminal e o tráfego capturado.

Na segunda parte do trabalho prático, exploramos conceitos de máscara de rede e sub-redes no protocolo IPv4. Para além disso, foram também explorados conceitos como o Sub-endereçamento IP. Esta segunda parte do trabalho revelou-se mais trabalhosa e exigente o que nos causou algumas dificuldades tendo de recorrer à ajuda da docente. No entanto, estas dificuldades foram ultrapassadas conseguindo dar continuidade ao trabalho.

Com a realização deste trabalho prático, conseguimos também aprimorar diversos conceitos teóricos e perceber o aproveitamento que podemos retirar de ferramentas como o Core e o Wireshark.

Concluindo, este trabalho foi bem-sucedido, permitindo-nos aprofundar e consolidar diversos conceitos importantes de Redes de Computadores. Conseguimos responder a todas as perguntas de forma satisfatória.

5 Abreviaturas

- IP - *Internet Protocol*;
- TTL - *Time to live*;
- RTT - *Round-trip time*;
- ICMP - *Internet Control Message Protocol*;
- MTU - *Maximum Transmission Unit*;
- MAC - *Media Access Control*.