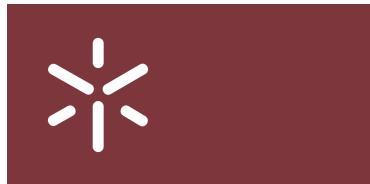


# Trabalho Prático N°2 - Protocolo IPv4 : Datagramas IP e Fragmentação

Hugo Rauber  
A104534

Aarón Vera  
E12045

Tomás Machado  
A104186



Universidade do Minho  
Escola de Engenharia  
Licenciatura em Engenharia Informática  
Unidade Curricular de Redes de Computadores

## Parte 1

### Exercício 1

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

#### Exercício 1.a

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

No.	Time	Source	Destination	Protocol	Length	Info
39	39.868573906	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=1/256, ttl=1 (no response..)
40	39.868575359	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=2/512, ttl=1 (no response..)
41	39.868576051	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=3/768, ttl=1 (no response..)
42	39.868576111	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=4/1024, ttl=2 (no response..)
43	39.868576332	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=5/1280, ttl=2 (no response..)
44	39.868579447	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=6/1536, ttl=2 (no response..)
45	39.868580198	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=7/1792, ttl=3 (no response..)
46	39.868581651	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=8/2048, ttl=3 (no response..)
47	39.868582853	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=9/2304, ttl=3 (no response..)
48	39.868584466	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=10/2560, ttl=4 (reply in ..)
49	39.868585859	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=11/2816, ttl=4 (reply in ..)
50	39.868586580	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=12/3072, ttl=4 (reply in ..)
51	39.868587412	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=13/3328, ttl=5 (reply in ..)
52	39.868588113	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=14/3584, ttl=5 (reply in ..)
53	39.868588654	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=15/3840, ttl=5 (reply in ..)
54	39.868589496	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=16/4096, ttl=6 (reply in ..)
55	39.898873697	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
56	39.898892916	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
57	39.898892644	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
58	39.900274948	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=17/4352, ttl=1 (reply in ..)
59	39.900284717	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=18/4608, ttl=6 (reply in ..)
60	39.900285232	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=19/4864, ttl=7 (reply in ..)
61	39.900290501	10.0.0.20	10.0.5.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
62	39.930300198	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
63	39.930301099	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
64	39.931191847	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=20/5120, ttl=7 (reply in ..)
65	39.931213393	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=21/5376, ttl=7 (reply in ..)
66	39.931219139	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=22/5632, ttl=8 (reply in ..)
67	39.956644896	10.0.2.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
68	39.960654013	10.0.2.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
69	39.960659063	10.0.2.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
70	39.961630133	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=23/5888, ttl=8 (reply in ..)
71	39.961644110	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=24/6144, ttl=8 (reply in ..)
72	39.961648979	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0018, seq=25/6400, ttl=9 (reply in ..)
74	40.021177663	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=10/2560, ttl=61 (request ..)
75	40.021181791	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=11/2816, ttl=61 (request ..)
76	40.021182693	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=12/3072, ttl=61 (request ..)
77	40.021183705	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=13/3328, ttl=61 (request ..)
78	40.021184859	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=14/3584, ttl=61 (request ..)
79	40.021185639	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=15/3840, ttl=61 (request ..)
80	40.021186971	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=16/4096, ttl=61 (request ..)
81	40.0212421690	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=17/4352, ttl=61 (request ..)
82	40.0212423973	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=18/4608, ttl=61 (request ..)
83	40.0212424945	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=19/4864, ttl=61 (request ..)
84	40.051757239	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=20/5120, ttl=61 (request ..)
85	40.051765074	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=21/5376, ttl=61 (request ..)
86	40.051766176	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=22/5632, ttl=61 (request ..)
87	40.082298922	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=23/5888, ttl=61 (request ..)
88	40.082307948	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=24/6144, ttl=61 (request ..)
89	40.082309902	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0018, seq=25/6400, ttl=61 (request ..)

Figura 1: Wireshark Lost -> Found (filtro icmp)

Na rede criada, capturamos os seguintes pacotes ICMP, gerados pelo comando traceroute -I, com o Wireshark no PC Lost. O traceroute enviou vários pacotes para o servidor Found 10.0.5.10. Foi possível ver que o ttl incrementa em 1 a cada 3 pacotes enviados. Isto é, começa com ttl=1 no pacote nº1, 2 e 3 e incrementa para ttl=2 nos pacotes nº4, 5 e 6 e assim por diante até ao 9. Este ttl será decrementado depois a cada nó atravessado, sendo eliminado quando chegar a 0 para evitar ficar com pacotes “sem destino”. Isto é retornado para a origem (como se pode ver na imagem nos pacotes a preto e verde) com a mensagem Time to live exceeded.

### Exercício 1.b

**Questão:** Qual deve ser o valor inicial mínimo do campo TTL para alcançar o servidor Found? Esboce um esquema com o valor do campo TTL à chegada a cada um dos routers percorridos até ao servidor Found. Verifique na prática que a sua resposta está correta.

O valor inicial mínimo do campo TTL deve ser 4. Isto porque podemos ver na figura 1, que as respostas do TTL exceeded aconteceram apenas 9 vezes. Isto é, resposta aos primeiros 3 pacotes com ttl=1, 3 pacotes com ttl=2 e outros 3 com ttl=3. Se não houve essa resposta ao ttl=4, quer dizer que foram recebidos no servidor Lost, com pelo menos um ttl=1. Assim, o valor mínimo é 4.

```

root@Lost:/tmp/pycore.43237/Lost.conf# ping -t 3 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
From 10.0.2.2 icmp_seq=1 Time to live exceeded
From 10.0.2.2 icmp_seq=2 Time to live exceeded
From 10.0.2.2 icmp_seq=3 Time to live exceeded
^Z
[1]+  Stopped                  ping -t 3 10.0.5.10
root@Lost:/tmp/pycore.43237/Lost.conf# ping -t 4 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=61 time=121 ms
^Z
[2]+  Stopped                  ping -t 4 10.0.5.10
root@Lost:/tmp/pycore.43237/Lost.conf# ping -t 5 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=61 time=121 ms
^Z
[3]+  Stopped                  ping -t 5 10.0.5.10
root@Lost:/tmp/pycore.43237/Lost.conf# █

```

Figura 2: Ping Lost -> Found

Para confirmar o resultado, efetuamos um ping de forma manual (**figura 2**) com ttl=3 (o qual se pode verificar que deu `time to live exceeded`), e ping ttl=4 e ttl=5 (para reforço) no qual se verificou que ambos chegaram ao destino.

### Exercício 1.c

**Questão:** 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`.

Através do envio de 10 pacotes de Lost para Found com o comando `ping -c 10 10.0.5.10`, recebemos estatísticas sobre esse envio, nas quais se observa o tempo médio pedido: 121.162 ms.

```

root@Lost:/tmp/pycore.43237/Lost.conf# ping -c 10 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=122 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=6 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=7 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=8 ttl=61 time=121 ms
64 bytes from 10.0.5.10: icmp_seq=9 ttl=61 time=122 ms
64 bytes from 10.0.5.10: icmp_seq=10 ttl=61 time=121 ms

--- 10.0.5.10 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 120.702/121.162/121.880/0.395 ms
root@Lost:/tmp/pycore.43237/Lost.conf#

```

Figura 3: RTT entre Lost e Found

### Exercício 1.d

**Questão:** O valor médio do atraso num sentido (One-Way Delay) poderia ser calculado com precisão dividindo o RTT por dois? O que torna difícil o cálculo desta métrica numa rede real?

Não, dividir o RTT por dois não seria viável, pois seria assumir que o tempo de ida e de volta são simétricos. Vários fatores, como:

- **Assimetria na Rota:** Os pacotes de ida e volta podem seguir caminhos diferentes, o que pode causar tempos de propagação distintos.
- **Variação no Tempo de Processamento:** Cada nó da rede pode demorar tempos diferentes para processar e encaminhar os pacotes.
- **Congestionamento da Rede:** Se houver filas nos routers, um pacote pode demorar mais tempo numa direção do que na outra.

## Exercício 2

**Questão:** Usando o wireshark capture o tráfego gerado pelo traceroute sem especificar o tamanho do pacote, i.e., quando é usado o tamanho do pacote de prova por defeito. Utilize como máquina destino o host marco.uminho.pt. Pare a captura. Com base no tráfego capturado, identifique os pedidos ICMP Echo Request e o conjunto de mensagens devolvidas como resposta. Selecione a primeira mensagem ICMP capturada e centre a análise no nível protocolar IP e, em particular, do cabeçalho IP (expanda o tab correspondente na janela de detalhe do wireshark).

### Exercício 2.a

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

Podemos ver que o endereço IP da interface ativa do computador utilizado (Source na imagem) é 192.168.1.24.

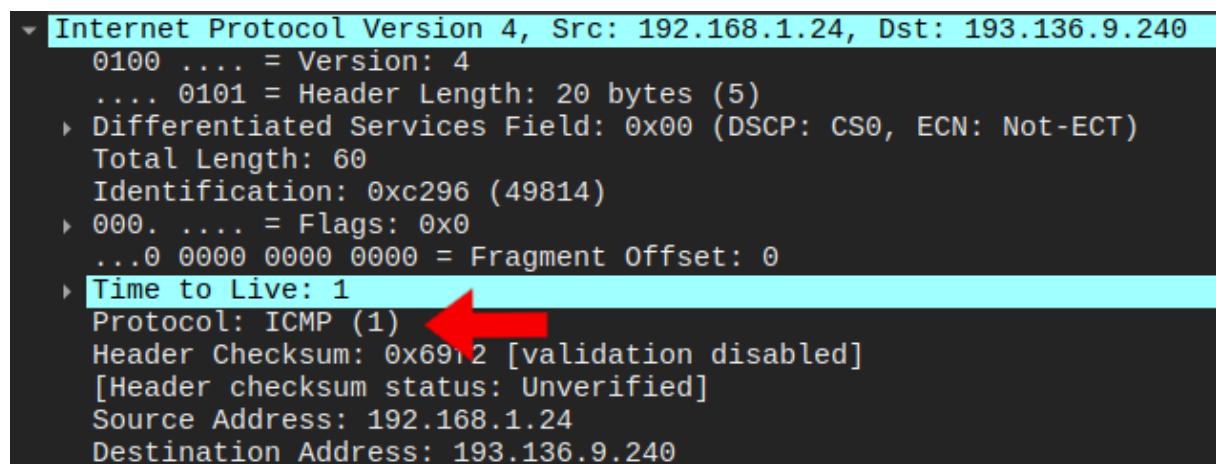
Source	Destination	Protocol	Length	Info
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=1/256, ttl=1 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=2/512, ttl=1 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=3/768, ttl=1 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=4/1024, ttl=2 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=5/1280, ttl=2 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=6/1536, ttl=2 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=7/1792, ttl=3 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=8/2048, ttl=3 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=9/2304, ttl=3 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=10/2560, ttl=4 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=11/2816, ttl=4 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=12/3072, ttl=4 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=13/3328, ttl=5 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=14/3584, ttl=5 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=15/3840, ttl=5 (no response found!)
192.168.1.24	193.136.9.240	ICMP	74	Echo (ping) request id=0x190f, seq=16/4096, ttl=6 (no response found!)

Figura 4: Requests feitos pela interface ativa pedida

### Exercício 2.b

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

O campo Protocol tem o valor 1, indicando que o pacote usa o **Internet Control Message Protocol (ICMP)**. Isso permite identificar que a mensagem capturada faz parte de um **Echo Request**, utilizado para diagnóstico de rede.



```
Internet Protocol Version 4, Src: 192.168.1.24, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xc296 (49814)
  ▶ 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
  ▶ Time to Live: 1
    Protocol: ICMP (1) ← Red arrow points here
    Header Checksum: 0x6912 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.24
    Destination Address: 193.136.9.240
```

Figura 5: Cabeçalho IP expandido (Protocol ICMP)

### Exercício 2.c

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

```

▼ Internet Protocol Version 4, Src: 192.168.1.24, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xc296 (49814)
  ▶ 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
  ▶ Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0x69f2 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.1.24
    Destination Address: 193.136.9.240

```

Figura 6: Cabeçalho IP expandido

O cabeçalho tem **20 bytes** como se pode ver na imagem. Para calcular o payload, retiramos à **Total Length** do pacote, o tamanho do cabeçalho **60 — 20**, o que nos deixa com um payload de **40 bytes**.

### Exercício 2.d

**Questão:** O datagrama IP foi fragmentado? Justifique.

```

▼ Internet Protocol Version 4, Src: 192.168.1.24, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xc296 (49814)
  ▶ 000. .... = Flags: 0x0
    0.... .... = Reserved bit: Not set
    .0... .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment Offset: 0
  ▶ Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0x69f2 [validation disabled]

```

Figura 7: Cabeçalho IP expandido (more fragments e fragment offset)

A partir dos campos **More fragments** e consequentemente **Fragment Offset**, determinamos que não foi fragmentado. Estando desativado o primeiro, querendo dizer que não houve fragmentação e, a 0 o segundo pois não houve nenhum fragmento anterior.

### Exercício 2.e

**Questão:** 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. Justifique estas mudanças.

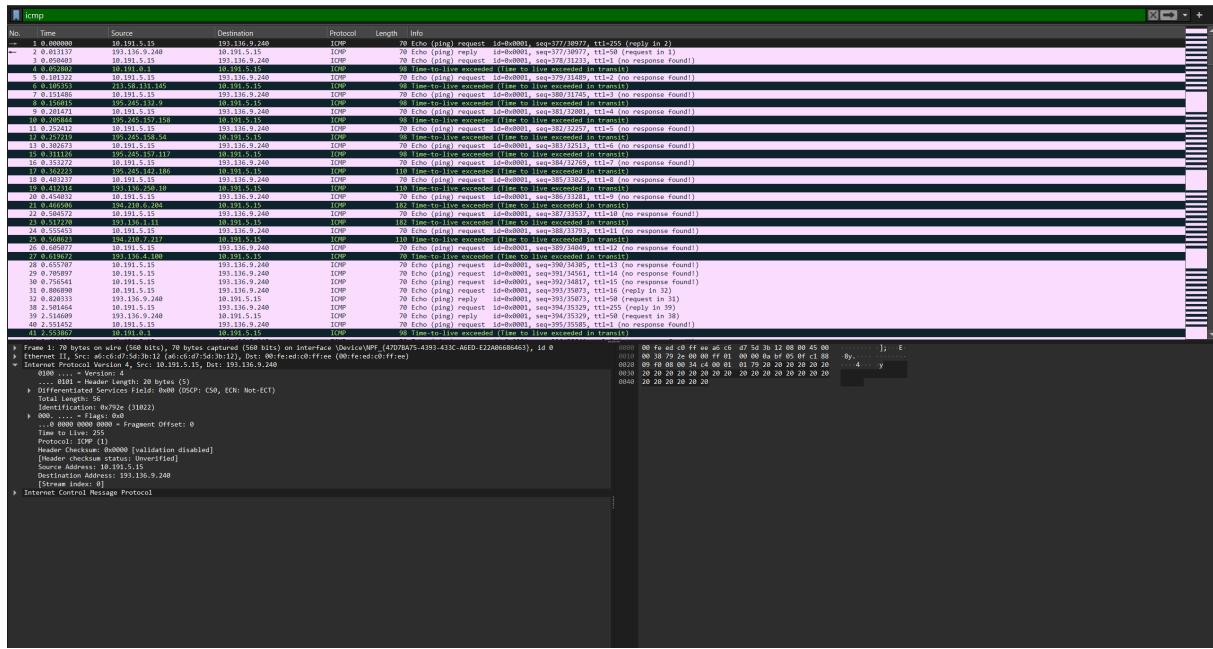


Figura 8: Wireshark Máquina Nativa (filtro icmp) -> host (marco.uminho.pt)

Como podemos observar pela figura 4, os campos do cabeçalho IP que variam de pacote para pacote são:

- O TTL é incrementado porque o traceroute aumenta o valor do TTL para cada novo pacote enviado até ao destino, garantindo que o pacote passe por diferentes routers e obtenha uma resposta de cada um deles;
- O Sequence Number (seq) é alterado porque a cada novo pacote ICMP enviado, um número de sequência diferente é atribuído para distinguir os pacotes e associar as respostas corretamente;

## Exercício 2.f

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

Como podemos observar na figura 4 tanto o datagrama IP e o TTL são incrementados por 1.

## Exercício 2.g

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

120 9.114566	172.26.20.116	193.136.9.240	ICMP	86 Echo (ping) request	id=0xb412, seq=10/2560, ttl=4 (reply in 121)
121 9.116841	193.136.9.240	172.26.20.116	ICMP	86 Echo (ping) reply	id=0xb412, seq=10/2560, ttl=61 (request in 120)
122 9.118597	172.26.20.116	193.136.9.240	ICMP	86 Echo (ping) request	id=0xb412, seq=11/2816, ttl=4 (reply in 123)
123 9.120481	193.136.9.240	172.26.20.116	ICMP	86 Echo (ping) reply	id=0xb412, seq=11/2816, ttl=61 (request in 122)
124 9.120646	172.26.20.116	193.136.9.240	ICMP	86 Echo (ping) request	id=0xb412, seq=12/3072, ttl=4 (reply in 125)
125 9.122501	193.136.9.240	172.26.20.116	ICMP	86 Echo (ping) reply	id=0xb412, seq=12/3072, ttl=61 (request in 124)

Figura 9: Tráfego ordenado

## Exercício 2.g.i

**Questão:** Qual é o valor do campo TTL recebido no seu computador? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL Exceeded recebidas no seu computador? Porquê?

O valor do campo TTL recebido no computador é 61, conforme observado nas mensagens de resposta ICMP Echo Reply capturadas no Wireshark.

Esse valor não permanece constante para todas as mensagens de resposta ICMP TTL Exceeded pois depende do número de saltos que o pacote percorreu antes de ser descartado. Cada router no caminho

reduz o TTL do pacote em 1 antes de encaminhá-lo. Assim, as mensagens ICMP TTL Exceeded são geradas em diferentes pontos da rota, resultando em variações no TTL quando chegam ao computador de origem.

Além disso, os routers que enviam as respostas ICMP TTL Exceeded configuram um valor TTL alto para garantir que essas respostas possam percorrer toda a rota de volta ao emissor sem serem descartadas prematuramente.

### Exercício 2.g.ii

**Questão:** Por que razão as mensagens de resposta ICMP TTL Exceeded são sempre enviadas na origem com um valor relativamente alto?

As mensagens de resposta ICMP TTL Exceeded são sempre enviadas na origem com um valor relativamente alto para garantir que consigam chegar ao computador de origem sem serem descartadas prematuramente.

Na captura fornecida, observa-se que o TTL nas respostas ICMP Echo Reply é 61, o que indica que os routers configuram um TTL alto ao enviar essas mensagens. Isso acontece porque, se o TTL fosse muito baixo, a resposta poderia ser descartada antes de chegar ao destino, impedindo que o emissor recebesse a notificação de que o TTL do pacote original foi excedido.

Dessa forma, os routers utilizam um TTL elevado para assegurar que a mensagem ICMP TTL Exceeded possa percorrer toda a rota de volta ao emissor, independentemente da quantidade de saltos envolvidos no trajeto.

### Exercício 2.h

**Questão:** A informação contida no cabeçalho ICMP poderia ser incluída no cabeçalho IPv4? Se sim, quais seriam as suas vantagens/desvantagens?

Como vantagem, incluir o ICMP no cabeçalho IPv4 reduziria a sobrecarga, eliminando a necessidade de pacotes separados, tornando o processamento mais rápido e eficiente. Isso poderia diminuir a latência e otimizar o controle do tráfego de rede.

A desvantagem é que modificar o IPv4 para incluir ICMP tornaria o protocolo mais complexo e poderia causar problemas de compatibilidade com dispositivos e redes existentes. Além disso, a separação atual permite maior flexibilidade, especialmente em redes que utilizam IPv4 e IPv6.

## Exercício 3

**Questão:** Pretende-se agora analisar a fragmentação de pacotes IP. Usando o wireshark, capture e observe o tráfego gerado depois do tamanho de pacote ter sido definido para  $(3800 + X)$  bytes, em que  $X$  é o número do grupo de trabalho (e.g.,  $X=22$  para o grupo PL22). De modo a poder visualizar os fragmentos, aceda a Edit -> Preferences -> Protocols e em IPv4 desative a opção “Reassemble fragmented IPv4 datagrams”.

O nosso  $X = 51$ , logo, utilizaremos 3851 bytes como tamanho de pacote.

### Exercício 3.a

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

14	6.782296432	172.26.23.214	193.136.9.240	ICMP	933 Echo (ping) request id=0x65fe, seq=1/256, ttl=64 (reply in 17)
15	6.789419599	193.136.9.240	172.26.23.214	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=4e42) [Reassembled in #17]
16	6.789492490	193.136.9.240	172.26.23.214	IPv4	933 Fragmented IP protocol (proto=ICMP 1, off=2960, ID=4e42) [Reassembled in #17]

Figura 10: Primeira imagem ICMP

Como podemos averiguar na figura 5, a fragmentação de pacotes é necessária quando um pacote é maior do que o MTU da interface de rede pelo qual ele está a ser transmitido. O MTU é o tamanho máximo de um pacote que pode ser transmitido pela interface de rede sem precisar ser dividido em partes menores, logo, este pacote deve ter excedido o MTU e teve de ser fragmentado em pedaços menores para ser transmitido com sucesso através da rede.

### **Exercício 3.b**

**Questão:** Imprima o primeiro fragmento do datagrama IP segmentado. 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?

```
▶ Frame 1: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{47D7BA75-4393-433C-A6ED-E22A06686463}, id 0
▶ Ethernet II, Src: 82:0e:2a:81:79:3c (82:0e:2a:81:79:3c), Dst: ComdaEnterpr_ff:94:00 (00:d0:03:ff:94:00)
└ Internet Protocol Version 4, Src: 172.26.95.190, Dst: 193.136.9.240
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCH: CS0, ECN: Not-ECT)
        Total Length: 1500
        Identification: 0x86c6 (34502)
    ▼ 001. .... = Flags: 0x1, More fragments
        0... .... = Reserved bit: Not set
        .0... .... = Don't fragment: Not set
        ..1.... = More fragments: Set
        ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.95.190
    Destination Address: 193.136.9.240
    [Stream index: 0]
```

Figura 11: Primeiro datagrama IP fragmentado

- Podemos ver que o datagrama foi fragmentado pois tem a flag More fragments ativa;
  - Como o Fragment Offset está a 0, podemos concluir que é o primeiro fragmento;

Figura 12: Tamanho do primeiro datagrama fragmentado

- Observando pela figura 7, o tamanho do primeiro datagrama IP é 1472 bytes (pois se utilizaram 8 bytes para identificar o ICMP e 20 para o header);

### Exercício 3.c

**Questão:** 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?

```

▶ Frame 2: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{47D7BA75-4393-433C-A6ED-E22A06686463}, id 0
▶ Ethernet II, Src: 82:0e:2a:81:79:3c (82:0e:2a:81:79:3c), Dst: ComdaEnterpr_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.95.190, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0x86c6 (34502)
  ▼ 001. .... = Flags: 0x1, More fragments
    0... .... = Reserved bit: Not set
    .0. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
    ...0 0000 1011 1001 = Fragment Offset: 1480
    Time to Live: 128
    Protocol: ICMP (1)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.95.190
    Destination Address: 193.136.9.240
    [Stream index: 0]
  ▼ Data (1480 bytes)
    Data [...] : 6162636465666768696a6b6c6d6e6f70717273747576776162636465666768696a6b6c6d6e6f70717273747576776162636465666768696a6b6c6d6e6f707172737475767
    [Length: 1480]

```

Figura 13: Tamanho do primeiro datagrama fragmentado

- Não se trata do primeiro fragmento pois temos a flag `Fragment Offset` a 1480 (tamanho do primeiro fragmento).
- Existem mais fragmentos, pois a flag `More fragments` mantém-se ativa.

### Exercício 3.d

**Questão:** Quantos fragmentos foram criados a partir do datagrama original? Como se detecta o último fragmento correspondente ao datagrama original? Estabeleça um filtro no Wireshark que permita listar apenas o último fragmento do primeiro datagrama IP segmentado.

```

▼ 000. .... = Flags: 0x0
  0.... .... = Reserved bit: Not set
  .0. .... = Don't fragment: Not set
  ..0. .... = More fragments: Not set
  ...0 0001 0111 0010 = Fragment Offset: 2960

```

Figura 14: Último datagrama

- Foram criados 3 fragmentos a partir do primeiro datagrama.
- Podemos detetar o último fragmento do datagrama quando a flag `More Fragments` a 0. Neste caso, podemos também ver que o `Fragment Offset` está a 2960 que é a soma dos tamanhos dos dois primeiros fragmentos.

ip.id==0x3a04 && ip.flags.mf == 0						
Time	Source	Destination	Protocol	Length	Info	
24 4.257282219	172.26.23.214	193.136.9.240	ICMP	933	Echo (ping) request id=0x6c08, seq=1/256, ttl=64 (reply in 27)	

Figura 15: Último datagrama com filtro

- Aqui filtramos o último fragmento com `ip.flags.mf == 0`, estabelecendo só que queremos fragmentos com a flag `More Fragments` a 0.

### Exercício 3.e

**Questão:** 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.

Como já podemos verificar nas figuras anteriores, os campos que são alterados no cabeçalho IP entre os diferentes fragmentos são:

- As flags que alteram entre `More Fragments` e `0x0`;
- O tamanho do fragment offset;
- O tamanho total dos pacotes;

### Exercício 3.f

**Questão:** Estime teoricamente o número de fragmentos gerados e o número de bytes transportados em cada um dos fragmentos. Apresente todos os cálculos efetuados, incluindo os campos do cabeçalho IP relevantes para cada um dos fragmentos.

Como o número do meu grupo é 51, o tamanho total seria 3.851 bytes

O MTU de uma rede é geralmente de 1.500 bytes. Este valor é o limite do tamanho dos pacotes IP que podem ser transmitidos sem necessidade de fragmentação. Calcule o número de fragmentos necessários: Para fragmentar um datagrama de 3.851 bytes, deve-se considerar que o tamanho de cada fragmento será de 1.500 bytes menos o tamanho do cabeçalho IP (20 bytes no IPv4). Cada pedaço terá um tamanho de 1.480 bytes ( $1.500 - 20$ ).

Para encontrar o número de fragmentos, dividimos o tamanho total dos bytes pelo tamanho de cada fragmento:  $3851/1480=3$

Isto indica que serão gerados 3 fragmentos.

Para calcular os bytes em cada pedaço:

O primeiro e o segundo blocos conterão 1.480 bytes de dados (mais os 20 bytes do cabeçalho IP), enquanto o terceiro bloco conterá os restantes (que serão inferiores a 1.480 bytes). Para o terceiro fragmento somamos os outros dois e subtraímos o número total de bytes:  $3851-(1480 \times 2)=891$

Fragmento 1: 1.500 bytes (1.480 dados + 20 cabeçalho). Fragmento 2: 1.500 bytes (1.480 dados + 20 cabeçalho). Fragmento 3: 911 bytes (891 dados + 20 cabeçalho).

### Exercício 3.g

**Questão:** Por que razão apenas o primeiro fragmento de cada pacote é identificado pelo Wireshark como sendo um pacote ICMP? Justifique a sua resposta com base no conceito de Fragmentação apresentado nas aulas teóricas.

O primeiro fragmento é o único que contém o cabeçalho ICMP completo, permitindo que seja identificado como um pacote ICMP pelo Wireshark. Os outros fragmentos são simplesmente porções do pacote original, sem informação ICMP completa.

### Exercício 3.h

**Questão:** 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 para decidir se deve ser fragmentado. Aumentar o MTU melhora a eficiência da transmissão, mas pode causar problemas de compatibilidade e perda de pacotes. Diminuir o MTU aumenta a compatibilidade, mas causa maior fragmentação e sobrecarga da rede.

### Exercício 3.i

**Questão:** 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.

Valor máximo de SIZE sem ocorrência de fragmentação: O valor máximo para o tamanho do pacote sem ocorrência de fragmentação é de 1480 bytes. Este é o tamanho máximo de dados que pode enviar num pacote sem ultrapassar o MTU de 1.500 bytes (que é comum nas redes Ethernet), considerando que o cabeçalho IP ocupa 20 bytes.

Justificação: O valor máximo de 1480 bytes é o tamanho dos dados que podem ser incluídos no pacote sem exceder o MTU de 1500 bytes. Quando é utilizada a opção DF (Don't Fragment), qualquer pacote que ultrapasse este tamanho será descartado, uma vez que não pode ser fragmentado.

```
aaron@MacBook-Pro-de-Aaron-Gomez-Vera ~ % ping -D -s 1480 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240): 1480 data bytes
```

## Parte 2

### Exercício 1

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

#### Exercício 1.a

**Questão:** De modo a garantir uma posição estratégicamente mais vantajosa e ter casa de férias para relaxar entre batalhas, ordena a construção de um segundo Castelo, em Braga. Não tendo qualquer queixa do serviço prestado, recorre aos serviços do ISP ReiDaNet, que já utiliza no condado, para ter acesso à rede no segundo Castelo. O ISP atribuiu-lhe o endereço de rede IP 172.68.XX.192/26 em que XX corresponde ao seu número de grupo (PLXX). Defina um esquema de endereçamento que permita o estabelecimento de pelo menos 6 redes e que garanta que cada uma destas possa ter 5 ou mais hosts. Assuma que todos os endereços de sub-redes são utilizáveis.

Para permitir pelo menos 6 redes com 5 ou mais hosts, partimos do endereço 172.68.51.192/26, onde 26 bits são fixos e 6 bits podem ser modificados.

Para identificar 6 redes, precisamos de 3 bits ( $2^3 = 8$ ), restando 3 bits para os hosts. O número de hosts por sub-rede é dado por  $2^3 - 2 = 6$ , satisfazendo o requisito mínimo.

Assim, o novo prefixo passa de /26 para /29, criando sub-redes de 8 endereços, com 6 utilizáveis. O intervalo original 172.68.51.192 - 172.68.51.255 é segmentado conforme a tabela:

SR	Prefixo Rede	Gama de Endereços Utilizáveis
000	172.68.51.192/29	172.68.51.193 → 172.68.51.198
001	172.68.51.200/29	172.68.51.201 → 172.68.51.206
010	172.68.51.208/29	172.68.51.209 → 172.68.51.214
011	172.68.51.216/29	172.68.51.217 → 172.68.51.222
100	172.68.51.224/29	172.68.51.225 → 172.68.51.230
101	172.68.51.232/29	172.68.51.233 → 172.68.51.238

Tabela 1: Tabela de Endereçamento

Dessa forma, garantimos 6 redes distintas, cada uma com 6 hosts disponíveis, atendendo aos requisitos.

#### Exercício 1.b

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

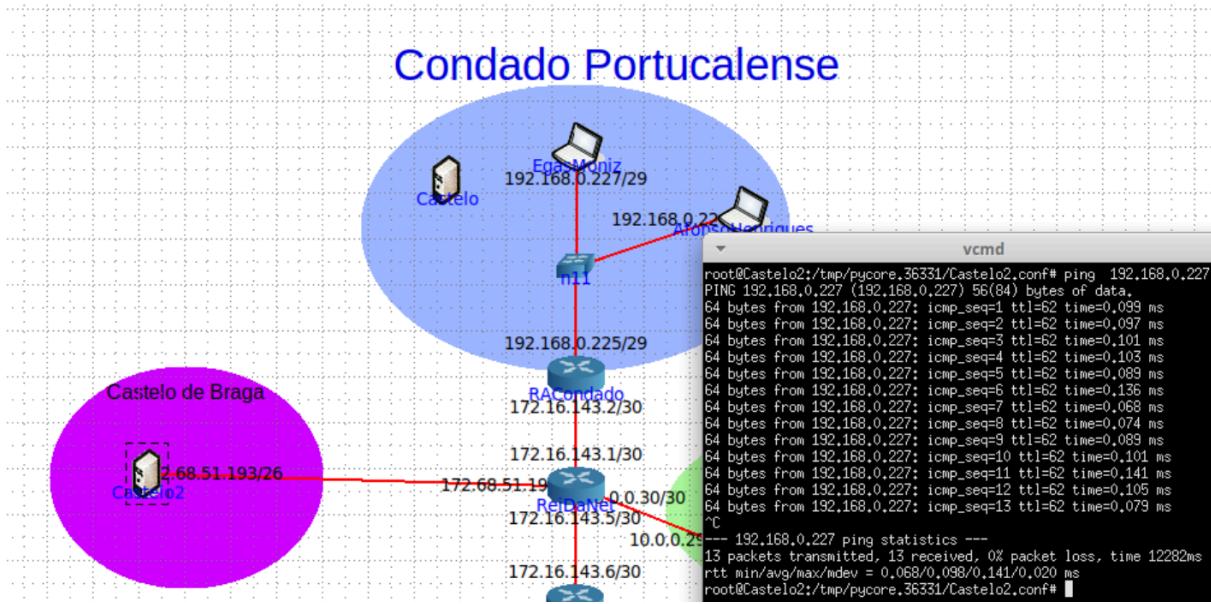


Figura 17: Verificação de conectividade entre Castelo2 e EgasMoniz

### Exercício 1.c

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

Primeiro, fez-se um `ip route del default` para eliminar as rotas default. Posteriormente, adicionou-se cada ip pretendido (Router do Condado e as 3 dentro da rede Institucional) de forma manual com `ip route add [ip da rede destino] via [ip router ReiDaNet] via eth0`. Assim, foram adicionadas as rotas para as redes. Com o comando `ping [IP destino]`, verificámos que as rotas estavam criadas e havia conexão entre elas.

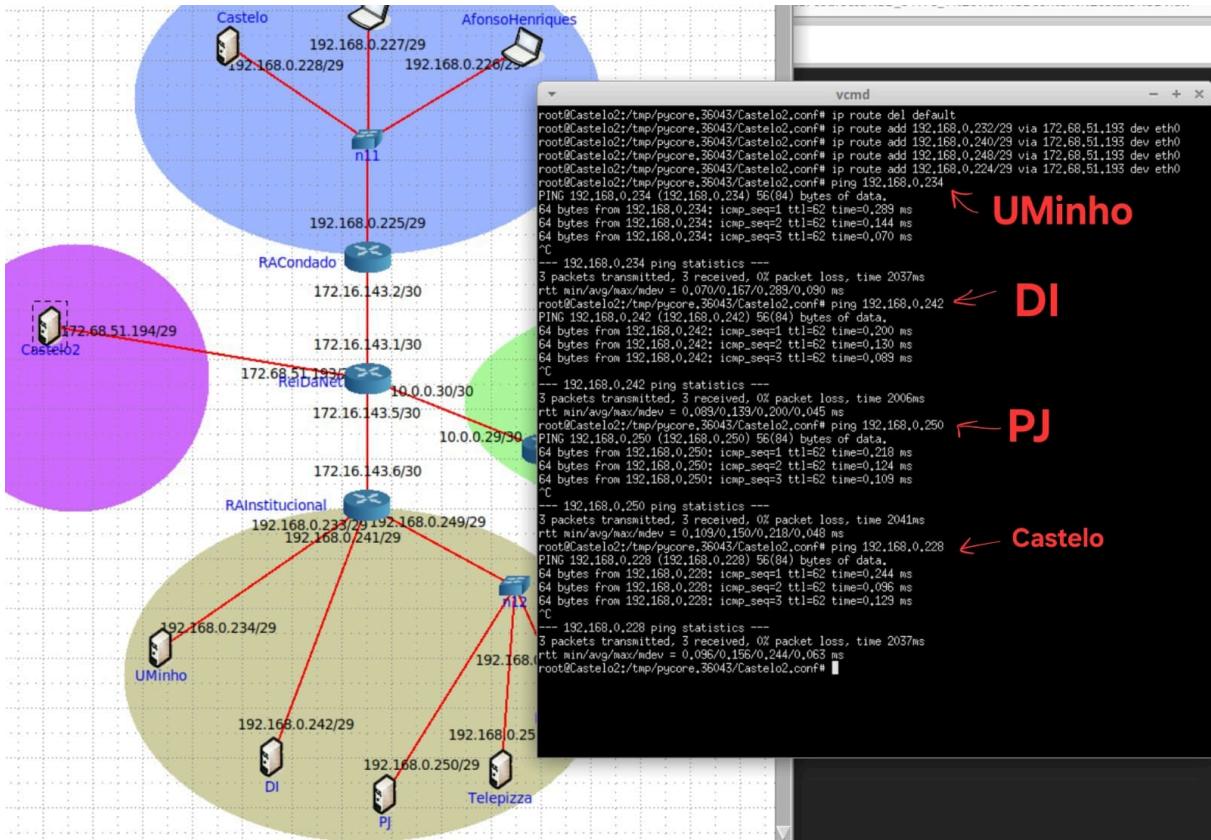


Figura 18: Novo acesso desde Castelo2 até ao Condado Portucalense e à rede Institucional

Quanto à rota default (0.0.0.0/0) é usada para encaminhar pacotes cujo destino não está definido em rotas mais específicas. Sem ela, pacotes destinados a redes externas ficarão sem caminho definido, resultando em perda de conectividade.

## Exercício 2

**Questão:** D.Afonso Henriques quer enviar fotos do novo Castelo à sua mãe, D.Teresa, mas está a ter alguns problemas de comunicação. Este alega que o problema deverá estar no dispositivo de D.Teresa, uma vez que no dia anterior conseguiu fazer stream de Fortnite para todos os seus subscritores da Twitch, e acabou de sair de uma discussão política no Reddit

### Exercício 2.a

**Questão:** Confirme, através do comando ping, que AfonsoHenriques tem efetivamente conectividade com os servidores Reddit e Twitch.

```

PING 192.168.0.145 (192.168.0.145) 56(84) bytes of data.
64 bytes from 192.168.0.145: icmp_seq=1 ttl=56 time=0.270 ms
64 bytes from 192.168.0.145: icmp_seq=2 ttl=56 time=0.147 ms
64 bytes from 192.168.0.145: icmp_seq=3 ttl=56 time=0.146 ms
64 bytes from 192.168.0.145: icmp_seq=4 ttl=56 time=0.285 ms
64 bytes from 192.168.0.145: icmp_seq=5 ttl=56 time=0.120 ms
64 bytes from 192.168.0.145: icmp_seq=6 ttl=56 time=0.142 ms

```

Figura 19: Verificação de conectividade entre AfonsoHenriques e os servidores Reddit

```

kycore.39885/AfonsoHenriques.conf# ping 192.168.0.138
PING 192.168.0.138 (192.168.0.138) 56(84) bytes of data.
64 bytes from 192.168.0.138: icmp_seq=1 ttl=55 time=0.238 ms
64 bytes from 192.168.0.138: icmp_seq=2 ttl=55 time=0.134 ms
64 bytes from 192.168.0.138: icmp_seq=3 ttl=55 time=0.160 ms
64 bytes from 192.168.0.138: icmp_seq=4 ttl=55 time=0.136 ms

```

Figura 20: Verificação de conectividade entre AfonsoHenriques e os servidores Twitch

Foi confirmado então que AfonsoHenriques tem conectividade com os servidores **Reddit** e **Twitch**.

### Exercício 2.b

**Questão:** 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@Teresa:/tmp/pycore.39885/Teresa.conf# netstat -rn
Kernel IP routing table
Destination      Gateway      Genmask      Flags      MSS Window irtt Iface
0.0.0.0          192.168.0.129 0.0.0.0      UG        0 0          0 eth0
192.168.0.128    0.0.0.0      255.255.255.248 U          0 0          0 eth0
root@Teresa:/tmp/pycore.39885/Teresa.conf#

```

Figura 21: Tabela de encaminhamento do dispositivo Teresa, mostrando a rota para a rede 192.168.0.128 com gateway ‘Gennak’ e interface ‘eth0’

```

root@AfonsoHenriques:/tmp/pycore.39885/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.39885/AfonsoHenriques.conf#

```

Figura 22: Tabla de enrutamiento del dispositivo AfonsoHenriques, que presenta una ruta incompleta hacia la red 192.168.0.128/29

Após a análise das tabelas de encaminhamento de ambos os dispositivos, AfonsoHenriques apresenta uma rota em falta para a rede 192.168.0.128/29, o que poderá estar a provocar a falta de conectividade com Teresa.

### Exercício 2.c

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

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

**n5:**

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	
192.168.0.0	10.0.0.30	255.255.255.0	UG	0	0	0	eth0	
192.168.0.128	10.0.0.25	255.255.255.248	UG	0	0	0	eth1	
192.168.0.136	10.0.0.25	255.255.255.248	UG	0	0	0	eth1	
192.168.0.144	10.0.0.25	255.255.255.248	UG	0	0	0	eth1	
192.168.0.152	10.0.0.25	255.255.255.248	UG	0	0	0	eth1	
192.168.0.224	10.0.0.30	255.255.255.248	UG	0	0	0	eth0	
192.168.0.232	10.0.0.30	255.255.255.248	UG	0	0	0	eth0	
192.168.0.240	10.0.0.30	255.255.255.248	UG	0	0	0	eth0	
192.168.0.248	10.0.0.30	255.255.255.248	UG	0	0	0	eth0	

Figura 23: Netstat de n5

route del -net 192.168.0.0 netmask 255.255.255.0

O comando remove a rota incorreta que devolia os pacotes para “trás”, em vez de os reencaminhar para n2.

**n2:**

vcmd								
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	
192.168.0.128	10.0.0.13	255.255.255.248	UG	0	0	0	eth1	
192.168.0.130	10.0.0.25	255.255.255.254	UG	0	0	0	eth2	
192.168.0.136	10.0.0.21	255.255.255.248	UG	0	0	0	eth0	
192.168.0.144	10.0.0.21	255.255.255.248	UG	0	0	0	eth0	
192.168.0.152	10.0.0.21	255.255.255.248	UG	0	0	0	eth0	
192.168.0.224	10.0.0.26	255.255.255.248	UG	0	0	0	eth2	
192.168.0.232	10.0.0.26	255.255.255.248	UG	0	0	0	eth2	
192.168.0.240	10.0.0.26	255.255.255.248	UG	0	0	0	eth2	
192.168.0.248	10.0.0.26	255.255.255.248	UG	0	0	0	eth2	

Figura 24: Netstat de n2

route del -net 192.168.0.130 netmask 255.255.255.254

O comando remove uma rota incorreta para 192.168.0.130, que estava configurada com a máscara 255.255.255.254. A rota errada fazia com que o tráfego para D.Teresa fosse enviado para um destino incorreto ou não saísse do n2.

**n1:**

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
10.0.0.0	10.0.0.9	255.255.255.252	UG	0	0	0	eth0
10.0.0.4	10.0.0.9	255.255.255.252	UG	0	0	0	eth0
10.0.0.8	0.0.0.0	255.255.255.252	U	0	0	0	eth0
10.0.0.12	0.0.0.0	255.255.255.252	U	0	0	0	eth1
10.0.0.16	10.0.0.9	255.255.255.252	UG	0	0	0	eth0
10.0.0.20	10.0.0.14	255.255.255.252	UG	0	0	0	eth1
10.0.0.24	10.0.0.14	255.255.255.252	UG	0	0	0	eth1
10.0.0.28	10.0.0.14	255.255.255.252	UG	0	0	0	eth1
172.0.0.0	10.0.0.14	255.0.0.0	UG	0	0	0	eth1
172.16.142.0	10.0.0.9	255.255.255.252	UG	0	0	0	eth0
172.16.142.4	10.0.0.9	255.255.255.252	UG	0	0	0	eth0
172.16.143.0	10.0.0.14	255.255.255.252	UG	0	0	0	eth1
172.16.143.4	10.0.0.14	255.255.255.252	UG	0	0	0	eth1
192.168.0.128	10.0.0.14	255.255.255.248	UG	0	0	0	eth1
192.168.0.136	10.0.0.9	255.255.255.248	UG	0	0	0	eth0
192.168.0.144	10.0.0.9	255.255.255.248	UG	0	0	0	eth0
192.168.0.152	10.0.0.9	255.255.255.248	UG	0	0	0	eth0
192.168.0.224	10.0.0.14	255.255.255.248	UG	0	0	0	eth1
192.168.0.232	10.0.0.14	255.255.255.248	UG	0	0	0	eth1
192.168.0.240	10.0.0.14	255.255.255.248	UG	0	0	0	eth1
192.168.0.248	10.0.0.14	255.255.255.248	UG	0	0	0	eth1

Figura 25: Netstat de n1

```
route add -net 192.168.0.128 netmask 255.255.255.248 gw 10.0.0.9  
Adiciona uma rota correta, redirecionando os pacotes para 10.0.0.9
```

n3:

The screenshot shows a terminal window titled "vcmd". The output displays the "Kernel IP routing table" with columns: Destination, Gateway, Genmask, Flags, MSS, Window, irtt, Iface. Numerous routes are listed, mostly for 192.168.0.x networks. At the bottom, a command is entered: "route add -net 192.168.0.128 netmask 255.255.255.248 gw 10.0.0.5". The prompt "root@n3:/tmp/pycore.44115/n3.conf# " is visible.

```
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
192.168.0.136   10.0.0.5      255.255.255.248 UG       0 0          0 eth2
192.168.0.144   10.0.0.5      255.255.255.248 UG       0 0          0 eth2
192.168.0.152   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
<3.conf# route add -net 192.168.0.128 netmask 255.255.255.248 gw 10.0.0.5
root@n3:/tmp/pycore.44115/n3.conf#
```

Figura 26: Netstat de n3 + comando

```
route add -net 192.168.0.128 netmask 255.255.255.248 gw 10.0.0.5
```

Adiciona uma rota no n3, indicando que os pacotes para 192.168.0.128/29. Isto, pois não existia uma rota que encaminhasse para Galiza.

## Exercício 2.d

**Questão:** 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.

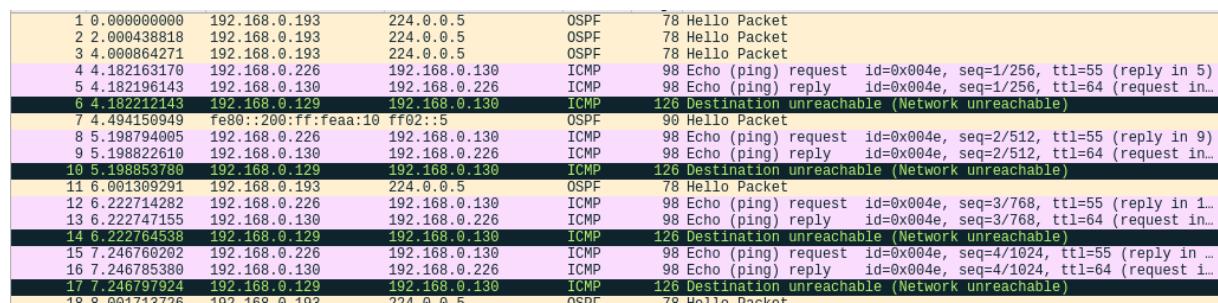


Figura 27: Wireshark de D.Teresa

Vemos que os pacotes enviados por Afonso Henriques são recebidos por D.Teresa, mas ao enviá-los, não consegue devolver uma resposta pois não há uma rota para tal.

## Exercício 2.d.i

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

```
vcmd
root@RAGaliza:/tmp/pycore.44115/RAGaliza.conf# traceroute 192.168.0.2225
192.168.0.2225: Temporary failure in name resolution
Cannot handle "host" cmdline arg `192.168.0.2225' on position 1 (argc 1)
root@RAGaliza:/tmp/pycore.44115/RAGaliza.conf# traceroute 192.168.0.225
traceroute to 192.168.0.225 (192.168.0.225), 30 hops max, 60 byte packets
connect: Network is unreachable
root@RAGaliza:/tmp/pycore.44115/RAGaliza.conf# []
```

Figura 28: Traceroute de RAGaliza para a Rede do Condado Portucalense

Vemos que não consegue encontrar uma rota para o Condado.

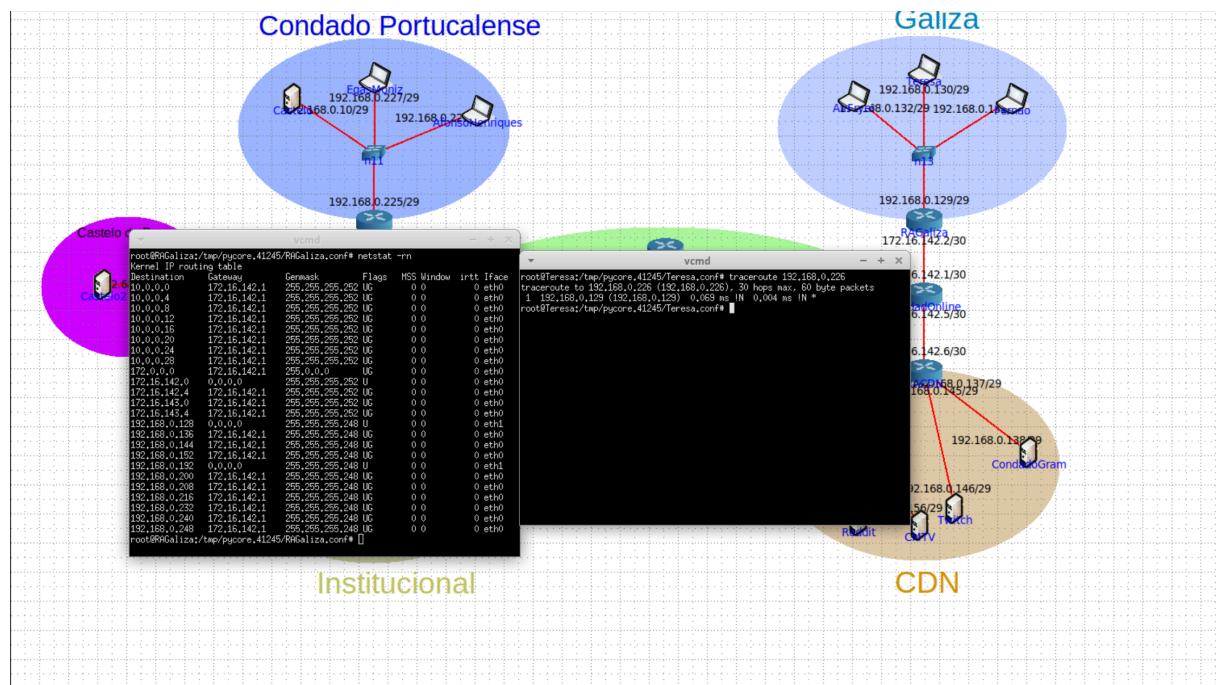


Figura 29: Tabela de routing de RAGaliza e traceroute de Teresa

Como podemos observar pela **figura 27** existe um problema de **recepção de resposta**. Teremos que adicionar, então, uma **rota de entrada de AfonsoHenriques para Teresa**.

4	4.001042927	192.168.0.193	224.0.0.5	OSPF	78 Hello Packet
5	4.826599461	192.168.0.226	192.168.0.130	ICMP	98 Echo (ping) request id=0x005e, seq=1/256, ttl=55 (r
6	4.826623766	192.168.0.130	192.168.0.226	ICMP	98 Echo (ping) reply id=0x005e, seq=1/256, ttl=64 (r
7	5.828011085	192.168.0.226	192.168.0.130	ICMP	98 Echo (ping) request id=0x005e, seq=2/512, ttl=55 (r
8	5.828036894	192.168.0.130	192.168.0.226	ICMP	98 Echo (ping) reply id=0x005e, seq=2/512, ttl=64 (r
9	6.002225760	192.168.0.193	224.0.0.5	OSPF	78 Hello Packet
10	6.842174400	192.168.0.226	192.168.0.130	ICMP	98 Echo (ping) request id=0x005e, seq=3/768, ttl=55 (r
11	6.842190390	192.168.0.130	192.168.0.226	ICMP	98 Echo (ping) reply id=0x005e, seq=3/768, ttl=64 (r
12	7.866075152	192.168.0.226	192.168.0.130	ICMP	98 Echo (ping) request id=0x005e, seq=4/1024, ttl=55 (r
13	7.866111030	192.168.0.130	192.168.0.226	ICMP	98 Echo (ping) reply id=0x005e, seq=4/1024, ttl=64 (r
14	8.002473621	192.168.0.193	224.0.0.5	OSPF	78 Hello Packet
15	8.889966283	192.168.0.226	192.168.0.130	ICMP	98 Echo (ping) request id=0x005e, seq=5/1280, ttl=55 (r
16	8.889984427	192.168.0.130	192.168.0.226	ICMP	98 Echo (ping) reply id=0x005e, seq=5/1280, ttl=64 (r

Figura 30: Wireshark D.Teresa após adição da rota

Adicionada a rota com route add -net 192.168.0.224 netmask 255.255.255.248 gw 172.16.142.1, podemos verificar que já existe reply da D.Teresa.

### Exercício 2.d.ii

**Questão:** 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

```
vcmd
<pycore.44115/AfonsoHenriques.conf# traceroute -I 192.168.0.130
traceroute to 192.168.0.130 (192.168.0.130), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.084 ms  0.013 ms  0.010 ms
 2  172.16.143.1 (172.16.143.1)  0.045 ms  0.015 ms  0.026 ms
 3  10.0.0.29 (10.0.0.29)  0.036 ms  0.019 ms  0.016 ms
 4  10.0.0.25 (10.0.0.25)  0.055 ms  0.036 ms  0.021 ms
 5  10.0.0.13 (10.0.0.13)  0.059 ms  0.037 ms  0.036 ms
 6  10.0.0.17 (10.0.0.17)  0.078 ms  0.175 ms  0.047 ms
 7  10.0.0.5 (10.0.0.5)  0.080 ms  0.049 ms  0.113 ms
 8  10.0.0.1 (10.0.0.1)  0.086 ms  0.053 ms  0.050 ms
 9  172.16.142.2 (172.16.142.2)  0.075 ms  0.046 ms  0.044 ms
10  192.168.0.130 (192.168.0.130)  0.095 ms  0.103 ms  0.076 ms
root@AfonsoHenriques:/tmp/pycore.44115/AfonsoHenriques.conf# 

vcmd
root@Teresa:/tmp/pycore.44115/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  192.168.0.129 (192.168.0.129)  0.065 ms  0.008 ms  0.007 ms
 2  172.16.142.1 (172.16.142.1)  0.033 ms  0.009 ms  0.009 ms
 3  10.0.0.2 (10.0.0.2)  0.033 ms  0.011 ms  0.010 ms
 4  10.0.0.6 (10.0.0.6)  0.035 ms  0.014 ms  0.049 ms
 5  10.0.0.18 (10.0.0.18)  0.038 ms  0.016 ms  0.016 ms
 6  10.0.0.14 (10.0.0.14)  0.045 ms  0.128 ms  0.028 ms
 7  10.0.0.26 (10.0.0.26)  0.038 ms  0.029 ms  0.019 ms
 8  10.0.0.30 (10.0.0.30)  0.044 ms  0.023 ms  0.030 ms
 9  172.16.143.2 (172.16.143.2)  0.051 ms  0.026 ms  0.033 ms
10  192.168.0.226 (192.168.0.226)  0.040 ms  0.028 ms  0.034 ms
root@Teresa:/tmp/pycore.44115/Teresa.conf# 
```

Figura 31: Traceroute D.Afonso-> D.Teresa e D.Teresa -> D.Afonso

Podemos ver que, de D.Afonso para D.Teresa, os pacotes passam por n1 (10.0.0.13) e quando voltam, passam por n4 (10.0.0.18).

## Exercício 2.e

**Questão:** Estando restabelecida a conectividade entre os dois hosts, obtenha a tabela de encaminhamento de n5 e foque-se na seguinte entrada:

```
ip route 192.168.0.0 255.255.255.0 10.0.0.30
```

Figura 32: Imagem enunciado

A entrada de rota ip route 192.168.0.0 255.255.255.0 10.0.0.30 cobre toda a rede 192.168.0.0/24 e encaminha pacotes para o próximo salto 10.0.0.30.

**Questão:** 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.

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
192.168.0.128	10.0.0.25	255.255.255.248	UG	0	0	0	eth1
192.168.0.136	10.0.0.25	255.255.255.248	UG	0	0	0	eth1
192.168.0.144	10.0.0.25	255.255.255.248	UG	0	0	0	eth1
192.168.0.152	10.0.0.25	255.255.255.248	UG	0	0	0	eth1
192.168.0.224	10.0.0.30	255.255.255.248	UG	0	0	0	eth0
192.168.0.232	10.0.0.30	255.255.255.248	UG	0	0	0	eth0
192.168.0.240	10.0.0.30	255.255.255.248	UG	0	0	0	eth0
192.168.0.248	10.0.0.30	255.255.255.248	UG	0	0	0	eth0

Figura 33: Tabela encaminhamento n5

Como há rotas mais específicas para a Galiza e a CDN, essa entrada não será utilizada para esses destinos. Se essa entrada estivesse na tabela, ela só seria usada se nenhuma das outras rotas existisse. Como temos as rotas detalhadas, o router sempre escolherá a mais específica.

## Exercício 2.f

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

Private address range		
Class	start address	finish address
A	10.0.0.0	10.255.255.255
B	172.16.0.0	172.31.255.255
C	192.168.0.0	192.168.255.255

Public address range		
Class	start address	finish address
A	0.0.0.0	126.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	254.255.255.255

Figura 34: Public/private address range

Pela tabela, verificamos que os endereços dos quatro polos são privados e do tipo C (192.168.x.x). Os utilizados na rede core, também são privados mas do tipo A (10.x.x.x).

### Exercício 2.g

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

Os switches de Layer 2 não possuem IP porque operam na camada de enlace, onde o encaminhamento de pacotes é feito com base nos endereços MAC. Como não realizam roteamento entre redes, não precisam de um endereço IP para funcionar.

### Exercício 3

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

### Exercício 3.a

**Questão:** 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.

## **Exercício 3.b**

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

## **Exercício 3.c**

**Questão:** Comente os aspectos positivos e negativos do uso do Supernetting.

### **-Aspectos positivos do supernetting:**

- Reduz a complexidade das tabelas de encaminhamento;
- Melhora a eficiência do uso de endereços IP;
- Otimiza o desempenho da rede ao minimizar a sobrecarga dos routers.

### **-Aspectos negativos do supernetting:**

- Dificuldade em expandir redes sem comprometer a agregação de rotas e a complexidade na implementação, exigindo assim planeamento para evitar conflitos e garantir compatibilidade com protocolos e dispositivos;
- Em redes muito grandes ou complexas, o supernetting pode atingir limitações de escalabilidade à medida que o número de sub-redes aumenta, pode se tornar mais difícil ou impraticável combinar todas em uma única rota supernet;