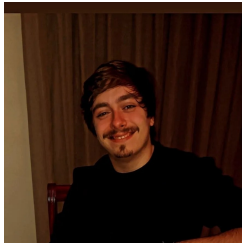


Redes de Computadores

Relatório Trabalho Prático 2

Protocolo IPv4 :: Datagramas IP e Fragmentação
Grupo 9 output obtido LEI - 2º Ano - 2º Semestre

Ano Letivo 2024/2025



Tiago Guedes
A97369



Diogo Goncalves
A101919



Tiago Carneiro
A93207

Braga,
4 de junho de 2025

Conteúdo

1	TP1	3
1.1	Exercicio 1	3
1.1.1	Alinea A	3
1.1.2	Observação dos pacotes no Wireshark	4
1.1.3	Alinea B	4
1.1.4	Alinea C	5
1.1.5	Alinea D	5
1.2	Exercicio 2	5
1.2.1	Alinea A	6
1.2.2	Alinea B	6
1.2.3	Alinea C	7
1.2.4	Alinea D	7
1.2.5	Alinea E	7
1.2.6	Alinea F	8
1.2.7	Alinea G	8
1.2.8	Alinea G I	8
1.2.9	Alinea G II	9
1.2.10	Alinea H	9
1.3	Exercicio 3	9
1.3.1	Alinea A	9
1.3.2	Alinea B	9
1.3.3	Alinea C	10
1.3.4	Alinea D	10
1.3.5	Alinea E	11
1.3.6	Alinea F	11
1.3.7	Alinea G	12
1.3.8	Alinea H	12
1.3.9	Alinea I	12
2	TP2	13
2.1	Exercicio 1	13
2.1.1	Alinea A	13
2.1.2	Alinea B	14
2.1.3	Alinea C	14
2.2	Exercicio 2	16
2.2.1	Alinea A	16
2.2.2	Alinea B	17
2.2.3	Alinea C	17
2.2.4	Alinea D	19
2.2.5	Alinea D I	19
2.2.6	Alinea D II	20
2.2.7	Alinea E	21
2.2.8	Alinea F	21
2.2.9	Alinea G	21
2.3	Exercicio 3	22
2.3.1	Alinea A	22
2.3.2	Alinea B	23
2.3.3	Alinea C	24

Lista de Figuras

1	Topologia	3
2	Traceroute 1.a	3
3	Wireshark Output	4
4	Traceroute 1 C	5
5	Ping Plotter	5
6	Questão 2 Wireshark	6
7	Questão 2 Internet Protocol	6
8	Questão 2 B Protocol	6
9	Questão 2 D Flag	7
10	Questão 2 G Ordered	8
11	Fragmentação do pacote	9
12	Detalhes da fragmentação do primeiro fragmento	10
13	Detalhes da fragmentação do segundo fragmento	10
14	Detalhes da fragmentação do terceiro fragmento	11
15	Topologia do Reino	13
16	Verificação de conectividade do Castelo 2 com os dispositivos do Condado Portu- galense	14
17	Apagar a rota default de Castelo 2	15
18	Adicionar rota para RDAInstitucional	15
19	Adicionar rota para Condado Portuagalense	16
20	Ping Afonso Henriques -> Twitch	16
21	Ping Afonso Henriques -> Reddit	16
22	NetStat AfonsoHenriques	17
23	NetStat Teresa	17
24	Ip Route Show de N3 e adição de rota para a rede 192.168.0.128/29	18
25	Em N2 foi apagada a route para 192.168.0.130/31	18
26	Troca de via 10.0.0.14 para 10.0.0.9	19
27	Conexão para 10.0.0.1 CondadOnline	19
28	Confirmação Wireshark	19
29	Teresa não consegue responder a AfonsoHenriques	20
30	Rota efetuada de Teresa para AfonsoHenriques	20
31	TraceRoute para Galiza	21
32	TraceRoute para CDN	21
33	Remover rotas para Galiza e CDN	22
34	Adicionar rota e executar um ping	22
35	Remover rotas para CondadoPortuagalense e Institucional	23
36	Adição de rotas supernet para CondadoPortuagalense e Institucional	23
37	Adição de rotas supernet para CondadoPortuagalense e Institucional	23

1 TP1

1.1 Exercício 1

1. Prepare uma topologia CORE para verificar o comportamento do `traceroute`. Na topologia deve existir: um *host* (pc) cliente designado *Lost*, cujo *router* de acesso é RA1; o *router* RA1 está simultaneamente ligado a dois *routers* na 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.

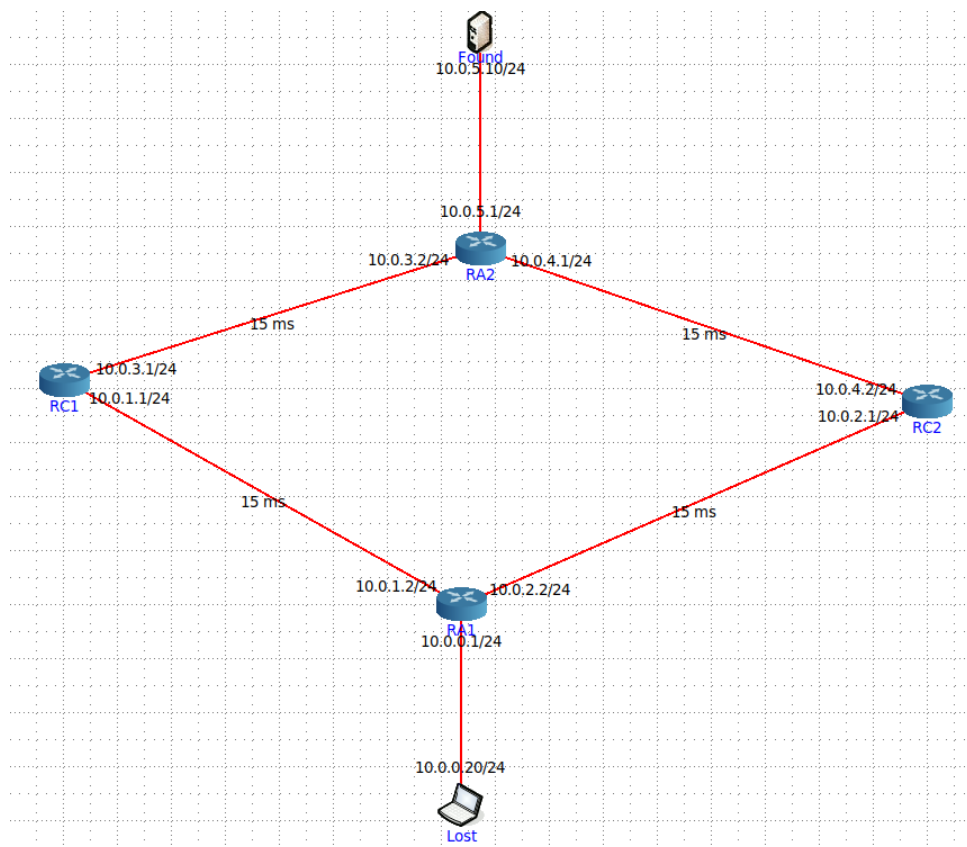


Figura 1: Topologia

1.1.1 Alinea A

- Active o Wireshark no *host* *Lost*. Numa *shell* de *Lost* execute o comando `traceroute -I 10.0.5.10` 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`.

Correndo o `traceroute` com o comando pedido:

```
root@Lost:/tmp/pycore.43575/Lost.conf# traceroute -I 10.0.5.10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.289 ms  0.011 ms  0.005 ms
 2  10.0.1.1 (10.0.1.1)  30.874 ms  30.866 ms  30.863 ms
 3  10.0.3.2 (10.0.3.2)  62.188 ms  62.186 ms  62.184 ms
 4  10.0.5.10 (10.0.5.10)  62.188 ms  62.186 ms  62.181 ms
root@Lost:/tmp/pycore.43575/Lost.conf#
```

Figura 2: Traceroute 1.a

Fomos capazes de obter o seguinte resultado no Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
29	25.098735897	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=1/256, ttl=1 (no response..
30	25.098750174	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
31	25.098757897	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=2/512, ttl=1 (no response..
32	25.098761607	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
33	25.098764607	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=3/768, ttl=1 (no response..
34	25.098768285	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
35	25.098770969	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=4/1024, ttl=2 (no respons..
36	25.098782953	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=5/1280, ttl=2 (no respons..
37	25.098785422	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=6/1536, ttl=2 (no respons..
38	25.098787852	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=7/1792, ttl=3 (no respons..
39	25.098791287	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=8/2048, ttl=3 (no respons..
40	25.098793421	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=9/2304, ttl=3 (no respons..
41	25.098795785	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=10/2560, ttl=4 (reply in ..
42	25.098797794	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=11/2816, ttl=4 (reply in ..
43	25.098799812	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=12/3072, ttl=4 (reply in ..
44	25.098802140	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=13/3328, ttl=5 (reply in ..
45	25.098804143	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=14/3584, ttl=5 (reply in ..
46	25.098806733	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=15/3840, ttl=5 (reply in ..
47	25.098809809	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=16/4096, ttl=6 (reply in ..
48	25.098830267	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=17/4352, ttl=6 (reply in ..
49	25.0988308299	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=18/4608, ttl=6 (reply in ..
50	25.098910931	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=19/4864, ttl=7 (reply in ..
51	25.160531980	10.0.1.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
52	25.160538331	10.0.1.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
53	25.160539412	10.0.1.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
54	25.161176603	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=20/5120, ttl=7 (reply in ..
55	25.161191760	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=21/5376, ttl=7 (reply in ..
56	25.161198106	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x001c, seq=22/5632, ttl=8 (reply in ..
57	25.192421798	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
58	25.192429382	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
59	25.192430377	10.0.3.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
60	25.192431299	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=10/2560, ttl=61 (request ..
61	25.192432293	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=11/2816, ttl=61 (request ..
62	25.192433198	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=12/3072, ttl=61 (request ..
63	25.192433980	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=13/3328, ttl=61 (request ..
64	25.192434853	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=14/3584, ttl=61 (request ..
65	25.192435719	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=15/3840, ttl=61 (request ..
66	25.192436578	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=16/4096, ttl=61 (request ..
67	25.192437436	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=17/4352, ttl=61 (request ..
68	25.192438388	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=18/4608, ttl=61 (request ..
69	25.192439242	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=19/4864, ttl=61 (request ..
70	25.224613325	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=20/5120, ttl=61 (request ..
71	25.224612151	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=21/5376, ttl=61 (request ..
72	25.224622251	10.0.5.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x001c, seq=22/5632, ttl=61 (request ..

Figura 3: Wireshark Output

1.1.2 Observação dos pacotes no Wireshark

A captura do Wireshark mostra pedidos ICMP Echo com TTLs crescentes, recebendo respostas Time-to-Live Exceeded dos routers intermédios (RA1, RC1, RC2, RA2) e Echo Replies do Found. O traceroute mapeia o caminho explorando a expiração do TTL.

A captura de tráfego ICMP no Wireshark mostra diferentes tipos de pacotes:

- Echo Request (ping request): Pacotes ICMP enviados pelo host Lost para o destino Found (10.0.5.10).
- Time Exceeded (TTL Exceeded): Respostas dos Routers intermediários quando o TTL dos pacotes ICMP chega a zero.
- Echo Reply (ping reply): Respostas do destino Found quando o pacote finalmente chega ao destino.

A sequência dos pacotes mostra que o traceroute -I usa pacotes ICMP Echo Request e recebe respostas ICMP Time Exceeded até alcançar o destino.

A partir do TTL 4 o Found começa a receber os pacotes, e a partir do TTL 61 o Lost começa a receber Echo Replies do Found.

1.1.3 Alinea B

- 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 mínimo deve ser 4, pois a partir da topologia podemos perceber que o pacote faz o seguinte percurso :

Lost (TTL=4) → RA1 (TTL=3) → RC1 (TTL=2) → RA2 (TTL=1) → Found (TTL=0, processa)

1.1.4 Alinea C

- 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.

```
root@Lost:/tmp/pycore.39299/Lost.conf# traceroute -I 10.0.5.10 -q 10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.065 ms 0.006 ms 0.004 ms 0.003 ms 0.004 ms 0.003
ms * * * *
 2 10.0.1.1 (10.0.1.1) 30.851 ms 30.841 ms 30.838 ms 30.835 ms 30.833 ms
30.832 ms * * * *
 3 10.0.3.2 (10.0.3.2) 61.722 ms 61.719 ms 63.705 ms 63.662 ms 63.652 ms
63.651 ms * * * *
 4 10.0.5.10 (10.0.5.10) 62.056 ms 62.055 ms 62.135 ms 62.120 ms 61.797 ms
61.777 ms 61.776 ms 61.773 ms 60.753 ms 60.737 ms
```

Figura 4: Traceroute 1 C

$$RTT = \frac{\sum_{i=0}^{10} (pacote_i)}{10} ms \equiv RTT = 61.6979 ms$$

1.1.5 Alinea D

- 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?

Não, pois o cálculo de **One-Way Delay** dessa forma revelaria-se inadequado e impreciso.

A imprecisão deve-se ao facto de que o tempo de ida e o tempo de volta que um pacote pode demorar varia com facilidade. Um exemplo que poderia levar a uma imprecisão seria caso houvesse um congestionamento da rede.

1.2 Exercício 2

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).

Decidimos utilizar o **Ping Plotter** para fazer o **traceroute** no sistema **Windows** :

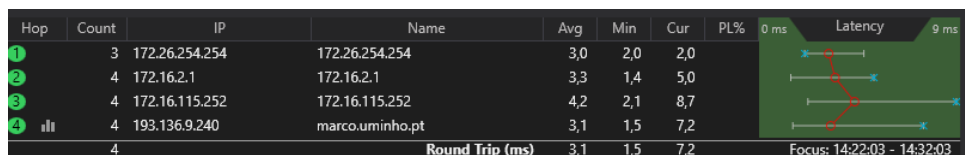


Figura 5: Ping Plotter

Devido a esta decisão observamos que seguindo os passos para a pergunta 2 obtivemos o seguinte no wireshark :

No.	Time	Source	Destination	Protocol	Length	Info
3	0.73435	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4752/36882, ttl=255 (reply in 4)
4	0.734432	193.136.9.240	172.26.42.241	ICMP	70	Echo (ping) reply id=0x0001, seq=4752/36882, ttl=61 (request in 3)
5	0.782429	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4753/37138, ttl=1 (no response found!)
6	0.783674	172.26.254.254	172.26.42.241	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
7	0.833257	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4754/37394, ttl=2 (no response found!)
8	0.838091	172.16.2.1	172.26.42.241	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
9	0.883831	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4755/37650, ttl=3 (no response found!)
10	0.885223	172.16.15.252	172.26.42.241	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
11	0.934983	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4756/37906, ttl=4 (reply in 12)
12	0.936348	193.136.9.240	172.26.42.241	ICMP	70	Echo (ping) reply id=0x0001, seq=4756/37906, ttl=61 (request in 11)

Figura 6: Questão 2 Wireshark

Sendo este o **Internet Protocol** da primeira mensagem ICMP capturada :

```

▼ Internet Protocol Version 4, Src: 172.26.42.241, 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: 56
    Identification: 0x867d (34429)
  ▶ 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0x92c3 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.42.241
    Destination Address: 193.136.9.240
    [Stream index: 1]
  ▶ Internet Control Message Protocol

```

Figura 7: Questão 2 Internet Protocol

1.2.1 Alinea A

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

Apartir do pacote capturado após ser efetuado o comando solicitado, conseguimos captar que o endereço IP da interface ativa do computador em que foi efetuado o comando é o mesmo que é representado como **Source**. Sendo assim, serão apartir dessa mesma **Source** que irão ser enviados os pacotes **ICMP**. Logo o endereço IP da interface ativa do nosso computador é 172.26.254.254.

1.2.2 Alinea B

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

Como podemos observar :

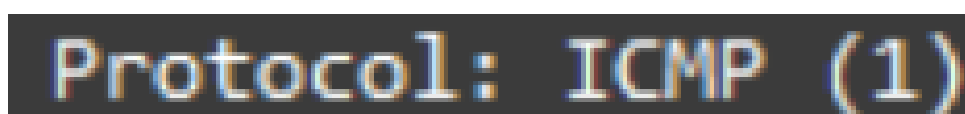


Figura 8: Questão 2 B Protocol

O valor do campo protocolo é 1, assim, permite-nos identificar que o tipo de pacote capturado é pertencente ao protocolo ICMP.

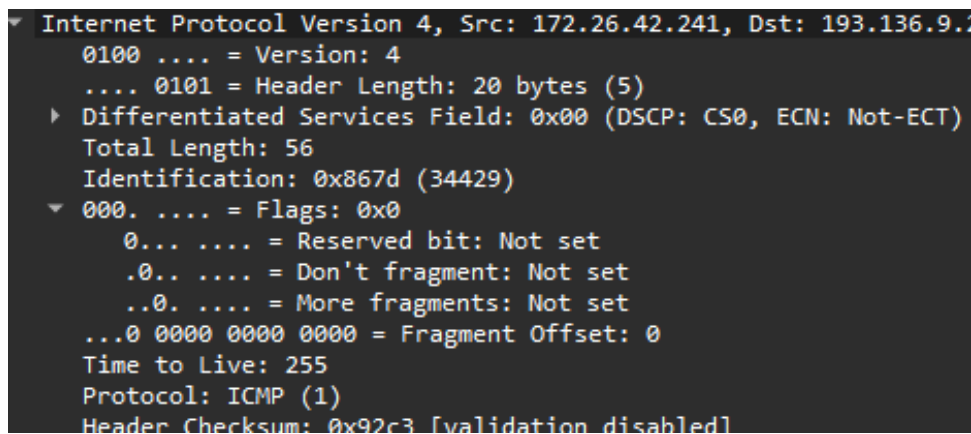
1.2.3 Alinea C

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

O cabeçalho IPv4 tem tamanho de 20 bytes, conforme podemos visualizar na figura anterior. Além disso é possível calcular o tamanho do campo de dados através da fórmula: $\text{payload} = \text{total length} - \text{header length}$ (1.2) Assim, temos que o valor do payload é de 40 bytes.

1.2.4 Alinea D

d. O datagrama IP foi fragmentado? Justifique.



```
Internet Protocol Version 4, Src: 172.26.42.241, Dst: 193.136.9.1
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x867d (34429)
  ▼ 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: 255
    Protocol: ICMP (1)
    Header Checksum: 0x92c3 [validation disabled]
```

Figura 9: Questão 2 D Flag

Não, o datagrama IP não foi fragmentado. Para verificarmos isto, basta verificar o campo **More Fragments** do datagrama e conferir que este indica que o pacote não possui qualquer fragmento. Além disso, outra forma de efetuarmos esta verificação seria observar o campo **Fragment Offset** que, no caso de um datagrama possuir fragmentos, existiria, obrigatoriamente, um pacote capturado com o campo diferente de 0.

1.2.5 Alinea E

- 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. Justifique estas mudanças.

Os campos do cabeçalho dos pacotes IP que variam são os seguintes:

- Identification - cada pacote IP terá associado a si um valor de identificação.
- Header Checksum - como o cabeçalho dos pacotes varia é também de esperar que o valor do seu checksum varie.
- Time to Live - do funcionamento do ICMP será de compreender que este valor irá variar.

A variação no TTL é um comportamento esperado em ferramentas como traceroute, pois permite mapear os saltos intermediários até o destino. A variação no ID do cabeçalho IP ocorre porque cada pacote enviado é tratado como um novo datagrama IP independente. O checksum muda devido às alterações nos valores dos campos variáveis do cabeçalho.

1.2.6 Alinea F

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

Sim, há um padrão nos valores de Identificação do datagrama IP e TTL:

Identificação do Datagrama IP (id=0x0001):

Todos os pacotes de Echo (ping) request possuem o mesmo valor de identificação (id=0x0001), indicando que são parte da mesma sequência de testes ICMP. Isso pode sugerir que a ferramenta utilizada para enviar os pacotes está fixando o campo de identificação, em vez de gerar valores únicos para cada datagrama. Time-to-Live (TTL):

O TTL nos pacotes Echo (ping) request começa com valores baixos e vai aumentando (ttl=1, ttl=2, ttl=3, ...). Isso indica que a captura foi feita durante um traceroute, onde cada pacote ICMP tem um TTL inicial baixo para mapear os routers ao longo do caminho. Quando o TTL do datagrama expira antes de atingir o destino, o roteador intermediário retorna um pacote "Time-to-live exceeded". Quando o destino final é alcançado, ele responde com um Echo (ping) reply.

1.2.7 Alinea G

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

Tráfego ordenado :

No.	Time	Source	Destination	Protocol	Length	Info
4	0.734432	193.136.9.240	172.26.42.241	ICMP	70	Echo (ping) reply id=0x0001, seq=4752/36882, ttl=61 (request in 3)
6	0.763674	172.26.42.254	172.26.42.241	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
8	0.838091	172.16.2.1	172.26.42.241	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10	0.885273	172.16.115.252	172.26.42.241	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
12	0.936348	193.136.9.240	172.26.42.241	ICMP	70	Echo (ping) reply id=0x0001, seq=4756/37906, ttl=61 (request in 11)
3	0.732435	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4752/36882, ttl=255 (reply in 4)
5	0.782429	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4753/37138, ttl=1 (no response found!)
7	0.833257	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4754/37394, ttl=2 (no response found!)
9	0.883831	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4755/37650, ttl=3 (no response found!)
11	0.934983	172.26.42.241	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=4756/37906, ttl=4 (reply in 12)

Figura 10: Questão 2 G Ordered

1.2.8 Alinea G I

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

O valor do campo TTL do primeiro pacote é de 255, do segundo 254 e do terceiro 253

Assim, podemos verificar que o valor não permanece constante nas mensagens de resposta ICMP TTL exceeded, isto deve-se ao facto do TTL diminuir em 1 unidade a cada salto.

É ainda interessante referir que cada router irá ter um valor de 256 para este campo em cada pacote que envia, no entanto, com os saltos presentes no caminho entre o router e a máquina destino, este valor diminui.

1.2.9 Alinea G II

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

As mensagens ICMP TTL *Exceeded* são enviadas pelos routers com um valor de TTL alto (normalmente 255) para garantir que elas consigam alcançar o destino sem expirar prematuramente.

1.2.10 Alinea H

- h. A informação contida no cabeçalho ICMP poderia ser incluída no cabeçalho IPv4? Se sim, quais seriam as suas vantagens/desvantagens?

Sim, mas aumentaria a complexidade do IPv4, dificultaria o processamento e reduziria a flexibilidade e a segurança. Manter o ICMP separado evita essas desvantagens e melhora a modularidade da rede.

1.3 Exercício 3

- 3. Pretende-se agora analisar a fragmentação de pacotes IP. Usando o Wireshark, capture e observe o tráfego gerado depois do tamanho de pacote ter sido definido para $(3800 + X)$ 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".

1.3.1 Alinea A

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

O pacote ICMP inicial foi gerado com o comando:

```
ping -s 3889 marco.uminho.pt
```

10 0.703271	172.26.42.241	193.136.9.240	ICMP	1514 Echo (ping) request id=0x0001, seq=4896/8211, ttl=1 (no response found!)
11 0.703271	172.26.42.241	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=906b)
12 0.703271	172.26.42.241	193.136.9.240	IPv4	943 Fragmented IP protocol (proto=ICMP 1, off=2960, ID=906b)

Figura 11: Fragmentação do pacote

A fragmentação foi necessária porque o tamanho total do datagrama (3917 bytes) excedeu o MTU da rede (1500 bytes).

1.3.2 Alinea B

- b. 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?

O primeiro fragmento tem as seguintes características:

```

▶ Frame 13: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{415731B6-C547-4...
▶ Ethernet II, Src: LiteonTechno_af:c4:3f (e0:0a:f6:af:c4:3f), Dst: ComdaEnterpr_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.42.241, 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: 0xe1fd (57853)
  ▼ 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: 255
    Protocol: ICMP (1)
    Header Checksum: 0x119f [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.42.241
    Destination Address: 193.136.9.240
    [Stream index: 5]
▶ Internet Control Message Protocol

```

Figura 12: Detalhes da fragmentação do primeiro fragmento

- **Tamanho Total:** 1514 bytes (incluindo cabeçalho Ethernet).
- **Flags:** More Fragments (MF) = 1.
- **Fragment Offset:** 0.

1.3.3 Alinea C

- 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?

O segundo fragmento tem as seguintes características:

```

▶ Frame 14: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{415731B6-C547-4...
▶ Ethernet II, Src: LiteonTechno_af:c4:3f (e0:0a:f6:af:c4:3f), Dst: ComdaEnterpr_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.42.241, 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: 0xe1fd (57853)
  ▼ 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: 255
    Protocol: ICMP (1)
    Header Checksum: 0x10e6 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.42.241
    Destination Address: 193.136.9.240
    [Stream index: 5]

```

Figura 13: Detalhes da fragmentação do segundo fragmento

- **Tamanho Total:** 1514 bytes.
- **Flags:** More Fragments (MF) = 1.
- **Fragment Offset:** 1480.

1.3.4 Alinea D

- d. 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.

Foram gerados 3 fragmentos. O último fragmento é identificado por:

```

▶ Frame 15: 943 bytes on wire (7544 bits), 943 bytes captured (7544 bits) on interface \Device\NPF_{41573186-C547-4861-8000-000000000000}
▶ Ethernet II, Src: LiteonTechno_af:c4:3f (e0:0a:f6:af:c4:3f), Dst: ComdaEnterpr_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.42.241, 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: 929
    Identification: 0xe1fd (57853)
  ▼ 000. .... = Flags: 0x0
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0001 0111 0010 = Fragment Offset: 2960
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0x3268 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.42.241
    Destination Address: 193.136.9.240
    [Stream index: 5]

```

Figura 14: Detalhes da fragmentação do terceiro fragmento

- **Flags:** More Fragments (MF) = 0.
- **Fragment Offset:** 2960

Filtro Wireshark para o último fragmento:

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

1.3.5 Alinea E

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

Os campos que mudam entre os fragmentos são:

- **Total Length:** o comprimento total do último fragmento pode ser diferente daquele dos fragmentos anteriores;
- **Fragment Offset:** o valor do deslocamento é específico para cada fragmento, o que possibilita a ordenação correta dos mesmos na reconstrução do datagrama original;
- **Header Checksum:** o checksum do cabeçalho varia, uma vez que os cabeçalhos dos fragmentos não são idênticos;
- **More Fragments:** a flag *More Fragments* encontra-se definida nos fragmentos que o precedem e não está definida no último, permitindo assim identificar qual o fragmento final.

1.3.6 Alinea F

- f. 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.

$$\sum_{i=1}^n (\text{size}(\text{fragment}_i) - \text{size}(\text{ip_header})) = \text{size}(\text{datagrama}) - \text{size}(\text{ip_header}) \quad (1)$$

Para o caso do traceroute com pacote de 3989:

$$(1500 - 20) + (1500 - 20) + (929 - 20) = 3889 - 20 \implies 3869 = 3869 \quad (2)$$

1.3.7 Alinea G

- g. 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 cabeçalho ICMP está presente apenas no primeiro fragmento. Os demais fragmentos contêm apenas partes dos dados.

1.3.8 Alinea H

- h. 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 ao MTU da rede. Alterar o MTU pode:

- **Aumentar MTU:** Reduz fragmentação, mas pode causar problemas em redes heterogêneas.
- **Diminuir MTU:** Aumenta fragmentação, mas evita problemas em redes com MTU menor.

1.3.9 Alinea I

- i. 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` = -1 (Windows) ou -s (Linux, Mac), determine o valor máximo de SIZE sem que ocorra fragmentação do pacote? Justifique o valor obtido.

O tamanho máximo de dados sem fragmentação é:

$$\text{MTU} - \text{Cabeçalho IP} - \text{Cabeçalho ICMP} = 1500 - 20 - 8 = 1472 \text{ bytes.}$$

Endereços utilizáveis :

$$2^3 - 2 = 6$$

Nova máscara de sub-rede: /29 (255.255.255.248).

Lista de sub-redes:

- Sub-rede 1: 172.68.89.192/29 (endereços: 172.68.89.193 a 172.68.89.198).
- Sub-rede 2: 172.68.89.200/29 (endereços: 172.68.89.201 a 172.68.89.206).
- Sub-rede 3: 172.68.89.208/29 (endereços: 172.68.89.209 a 172.68.89.214).
- Sub-rede 4: 172.68.89.216/29 (endereços: 172.68.89.217 a 172.68.89.222).
- Sub-rede 5: 172.68.89.224/29 (endereços: 172.68.89.225 a 172.68.89.230).
- Sub-rede 6: 172.68.89.232/29 (endereços: 172.68.89.233 a 172.68.89.238).

2.1.2 Alinea B

- b. Ligue um novo *host* Castelo2 diretamente ao *router* ReiDaNet. Associe-lhe um endereço, à sua escolha, pertencente a uma sub-rede disponível das criadas na alinea 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.

Configuração do Castelo2:

- Sub-rede escolhida: 172.68.89.192/29.
- Interface do router ReiDaNet: 172.68.89.193 (primeiro endereço válido).
- Endereço do Castelo2: 172.68.89.194.

Verificação de conectividade:

```
root@Castelo2:/tmp/pycore.35097/Castelo2.conf# ping 192.168.0.228
PING 192.168.0.228 (192.168.0.228) 56(84) bytes of data.
64 bytes from 192.168.0.228: icmp_seq=1 ttl=62 time=0.083 ms
64 bytes from 192.168.0.228: icmp_seq=2 ttl=62 time=0.060 ms
64 bytes from 192.168.0.228: icmp_seq=3 ttl=62 time=0.066 ms
64 bytes from 192.168.0.228: icmp_seq=4 ttl=62 time=0.065 ms
64 bytes from 192.168.0.228: icmp_seq=5 ttl=62 time=0.064 ms
```

Figura 16: Verificação de conectividade do Castelo 2 com os dispositivos do Condado Portucalense

2.1.3 Alinea C

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

Apagar a rota default de Castelo2:

```

root@Castelo2:/tmp/pycore.35097/Castelo2.conf# ip route show
default via 172.68.89.193 dev eth0
172.68.89.192/29 dev eth0 proto kernel scope link src 172.68.89.194
root@Castelo2:/tmp/pycore.35097/Castelo2.conf# ip route del default
root@Castelo2:/tmp/pycore.35097/Castelo2.conf# ip route show
172.68.89.192/29 dev eth0 proto kernel scope link src 172.68.89.194
root@Castelo2:/tmp/pycore.35097/Castelo2.conf# █

```

Figura 17: Apagar a rota default de Castelo 2

Adicionar rota para RDAInstitucional:

```

root@Castelo2:/tmp/pycore.35097/Castelo2.conf# ip route add 192.168.0.232/29 v>
root@Castelo2:/tmp/pycore.35097/Castelo2.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=62 time=0.120 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=62 time=0.056 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=62 time=0.058 ms
^C

```

Figura 18: Adicionar rota para RDAInstitucional

Usa-se 192.168.0.232/29 porque:

- 192.168.0.233/29 está dentro de 192.168.0.232/29
- 192.168.0.241/29 está dentro de 192.168.0.232/29
- 192.168.0.249/29 está dentro de 192.168.0.232/29

Adicionar rota para Condado Portucalense:

```
root@Castelo2:/tmp/pycore.35097/Castelo2.conf# ip route add 192.168.0.224/29 via 172.68.89.193
root@Castelo2:/tmp/pycore.35097/Castelo2.conf# ping 192.168.0.228
PING 192.168.0.228 (192.168.0.228) 56(84) bytes of data:
64 bytes from 192.168.0.228: icmp_seq=1 ttl=62 time=0.084 ms
64 bytes from 192.168.0.228: icmp_seq=2 ttl=62 time=0.057 ms
64 bytes from 192.168.0.228: icmp_seq=3 ttl=62 time=0.060 ms
64 bytes from 192.168.0.228: icmp_seq=4 ttl=62 time=0.056 ms
```

Figura 19: Adicionar rota para Condado Portucalense

Utilidade da rota default: Encaminha pacotes para destinos não listados nas rotas estáticas, garantindo conectividade geral.

2.2 Exercício 2

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

2.2.1 Alinea A

- a. Confirme, através do comando ping, que AfonsoHenriques tem efetivamente conectividade com os servidores Reddit e Twitch.

```
<ycore.35097/AfonsoHenriques.conf# ping 192.168.0.146
PING 192.168.0.146 (192.168.0.146) 56(84) bytes of data:
64 bytes from 192.168.0.146: icmp_seq=1 ttl=55 time=0.260 ms
64 bytes from 192.168.0.146: icmp_seq=2 ttl=55 time=0.110 ms
64 bytes from 192.168.0.146: icmp_seq=3 ttl=55 time=0.116 ms
64 bytes from 192.168.0.146: icmp_seq=4 ttl=55 time=0.106 ms
^C
```

Figura 20: Ping Afonso Henriques -> Twitch

```
PING 192.168.0.155 (192.168.0.155) 56(84) bytes of data:
64 bytes from 192.168.0.155: icmp_seq=1 ttl=55 time=0.116 ms
64 bytes from 192.168.0.155: icmp_seq=2 ttl=55 time=0.113 ms
64 bytes from 192.168.0.155: icmp_seq=3 ttl=55 time=0.107 ms
64 bytes from 192.168.0.155: icmp_seq=4 ttl=55 time=0.108 ms
64 bytes from 192.168.0.155: icmp_seq=5 ttl=55 time=0.109 ms
```

Figura 21: Ping Afonso Henriques -> Reddit

2.2.2 Alinea B

- 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.35097/AfonsoHenriques.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 192.168.0.225 0.0.0.0 UG 0 0 0 eth0
192.168.0.224 0.0.0.0 255.255.255.248 U 0 0 0 eth0
```

Figura 22: NetStat AfonsoHenriques

```
root@Teresa:/tmp/pycore.35097/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
```

Figura 23: NetStat Teresa

Não existe nenhum problema aparente com as entradas de nenhum dos dispositivos. Abaixo conseguimos visualizar os detalhes e a utilidade de cada entrada de cada um dos mesmos.

AfonsoHenriques :

- Rota Padrão (0.0.0.0 → 192.168.0.225, eth0): Envia tráfego para redes externas através do gateway 192.168.0.225, permitindo acesso à internet ou outras redes.
- Rota Local (192.168.0.224/29 → eth0): Permite comunicação direta com dispositivos na sub-rede 192.168.0.224 a 192.168.0.231, como o gateway.

Teresa :

- Rota Padrão (0.0.0.0 → 192.168.0.129, eth0): Direciona tráfego para redes externas via gateway 192.168.0.129, para acesso à internet ou outras redes.
- Rota Local (192.168.0.128/28 → eth0): Permite comunicação direta com dispositivos na sub-rede 192.168.0.128 a 192.168.0.143, como o gateway.

2.2.3 Alinea C

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

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

Começamos por correr o `ip route show` seguido de adicionar a rota para a rede 192.16.8.0.128/29:

```

root@n3:/tmp/pycore.35097/n3.conf# ip route show
10.0.0.0/30 via 10.0.0.5 dev eth2 proto zebra
10.0.0.4/30 dev eth2 proto kernel scope link src 10.0.0.6
10.0.0.8/30 dev eth0 proto kernel scope link src 10.0.0.9
10.0.0.12/30 via 10.0.0.10 dev eth0 proto zebra
10.0.0.16/30 dev eth1 proto kernel scope link src 10.0.0.17
10.0.0.20/30 via 10.0.0.18 dev eth1 proto zebra
10.0.0.24/30 via 10.0.0.18 dev eth1 proto zebra
10.0.0.28/30 via 10.0.0.10 dev eth0 proto zebra
172.0.0.0/8 via 10.0.0.10 dev eth0 proto zebra
172.16.142.0/30 via 10.0.0.5 dev eth2 proto zebra
172.16.142.4/30 via 10.0.0.5 dev eth2 proto zebra
172.16.143.0/30 via 10.0.0.18 dev eth1 proto zebra
172.16.143.4/30 via 10.0.0.10 dev eth0 proto zebra
192.168.0.136/29 via 10.0.0.5 dev eth2 proto zebra
192.168.0.144/29 via 10.0.0.5 dev eth2 proto zebra
192.168.0.152/29 via 10.0.0.5 dev eth2 proto zebra
192.168.0.224/29 via 10.0.0.18 dev eth1 proto zebra
192.168.0.232/29 via 10.0.0.10 dev eth0 proto zebra
192.168.0.240/29 via 10.0.0.10 dev eth0 proto zebra
192.168.0.248/29 via 10.0.0.10 dev eth0 proto zebra
root@n3:/tmp/pycore.35097/n3.conf# ip route add 192.168.0.128/29 via 10.0.0.5
root@n3:/tmp/pycore.35097/n3.conf# █

```

Figura 24: Ip Route Show de N3 e adição de rota para a rede 192.168.0.128/29

Após isso, eliminamos a rota para 192.168.0.13/31 :

```

10.0.0.0/30 via 10.0.0.13 dev eth1 proto zebra
10.0.0.4/30 via 10.0.0.21 dev eth0 proto zebra
10.0.0.8/30 via 10.0.0.13 dev eth1 proto zebra
10.0.0.12/30 dev eth1 proto kernel scope link src 10.0.0.14
10.0.0.16/30 via 10.0.0.13 dev eth1 proto zebra
10.0.0.20/30 dev eth0 proto kernel scope link src 10.0.0.22
10.0.0.24/30 dev eth2 proto kernel scope link src 10.0.0.25
10.0.0.28/30 via 10.0.0.26 dev eth2 proto zebra
172.0.0.0/8 via 10.0.0.26 dev eth2 proto zebra
172.16.142.0/30 via 10.0.0.13 dev eth1 proto zebra
172.16.142.4/30 via 10.0.0.21 dev eth0 proto zebra
172.16.143.0/30 via 10.0.0.26 dev eth2 proto zebra
172.16.143.4/30 via 10.0.0.26 dev eth2 proto zebra
192.168.0.128/29 via 10.0.0.13 dev eth1 proto zebra
192.168.0.130/31 via 10.0.0.25 dev eth2 proto zebra
192.168.0.136/29 via 10.0.0.21 dev eth0 proto zebra
192.168.0.144/29 via 10.0.0.21 dev eth0 proto zebra
192.168.0.152/29 via 10.0.0.21 dev eth0 proto zebra
192.168.0.224/29 via 10.0.0.26 dev eth2 proto zebra
192.168.0.232/29 via 10.0.0.26 dev eth2 proto zebra
192.168.0.240/29 via 10.0.0.26 dev eth2 proto zebra
192.168.0.248/29 via 10.0.0.26 dev eth2 proto zebra
root@n2:/tmp/pycore.35097/n2.conf# ip route del 192.168.0.130/31
root@n2:/tmp/pycore.35097/n2.conf# █

```

Figura 25: Em N2 foi apagada a route para 192.168.0.130/31

De seguida, trocamos a via 10.0.0.14 para a via 10.0.0.9:

```

10.0.0.8/30 dev eth0 proto kernel scope link src 10.0.0.10
10.0.0.12/30 dev eth1 proto kernel scope link src 10.0.0.13
10.0.0.16/30 via 10.0.0.9 dev eth0 proto zebra
10.0.0.20/30 via 10.0.0.14 dev eth1 proto zebra
10.0.0.24/30 via 10.0.0.14 dev eth1 proto zebra
10.0.0.28/30 via 10.0.0.14 dev eth1 proto zebra
172.0.0.0/8 via 10.0.0.14 dev eth1 proto zebra
172.16.142.0/30 via 10.0.0.9 dev eth0 proto zebra
172.16.142.4/30 via 10.0.0.9 dev eth0 proto zebra
172.16.143.0/30 via 10.0.0.14 dev eth1 proto zebra
172.16.143.4/30 via 10.0.0.14 dev eth1 proto zebra
192.168.0.128/29 via 10.0.0.14 dev eth1 proto zebra
192.168.0.136/29 via 10.0.0.9 dev eth0 proto zebra
192.168.0.144/29 via 10.0.0.9 dev eth0 proto zebra
192.168.0.152/29 via 10.0.0.9 dev eth0 proto zebra
192.168.0.224/29 via 10.0.0.14 dev eth1 proto zebra
192.168.0.232/29 via 10.0.0.14 dev eth1 proto zebra
192.168.0.240/29 via 10.0.0.14 dev eth1 proto zebra
192.168.0.248/29 via 10.0.0.14 dev eth1 proto zebra
root@n1:/tmp/pycore.35097/n1.conf# ip route del 192.168.0.128
RTNETLINK answers: No such process
root@n1:/tmp/pycore.35097/n1.conf# ip route del 192.168.0.128/29
root@n1:/tmp/pycore.35097/n1.conf# ip route add 192.168.0.128/29 via 10.0.0.9
root@n1:/tmp/pycore.35097/n1.conf#

```

Figura 26: Troca de via 10.0.0.14 para 10.0.0.9

Por fim, fazemos a conexão para 10.0.0.1:

```

<7/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) 0.029 ms 0.003 ms 0.002 ms
 2 172.16.143.1 (172.16.143.1) 0.012 ms 0.003 ms 0.003 ms
 3 10.0.0.29 (10.0.0.29) 0.014 ms 0.004 ms 0.003 ms
 4 10.0.0.25 (10.0.0.25) 0.013 ms 0.005 ms 0.005 ms
 5 10.0.0.13 (10.0.0.13) 0.013 ms 0.006 ms 0.005 ms
 6 10.0.0.17 (10.0.0.17) 0.019 ms 0.021 ms 0.006 ms
 7 10.0.0.5 (10.0.0.5) 0.015 ms 0.007 ms 0.007 ms
 8 10.0.0.1 (10.0.0.1) 0.014 ms 0.008 ms 0.008 ms
 9 * * *
10 * * *
11 * * *

```

Figura 27: Conexão para 10.0.0.1 CondadOnline

2.2.4 Alinea D

- 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.

No.	Time	Source	Destination	Protocol	Length	Info
9	12.233758589	192.168.0.226	192.168.0.130	ICMP	98	Echo (ping) request id=0x006c, seq=1/256, ttl=55 (reply in 1
10	12.233767999	192.168.0.130	192.168.0.226	ICMP	98	Echo (ping) reply id=0x006c, seq=1/256, ttl=64 (request in
11	12.233774453	192.168.0.129	192.168.0.130	ICMP	126	Destination unreachable (Network unreachable)

Figura 28: Confirmação Wireshark

2.2.5 Alinea D I

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

Não é possível para Teresa responder porque o router RAGaliza não tem nem rota padrão definida nem uma rota para o dispositivo AfonsoHenriques.

```
root@RAGaliza:/tmp/pycore.35097/RAGaliza.conf# ip route show
10.0.0.0/30 via 172.16.142.1 dev eth0 proto zebra
10.0.0.4/30 via 172.16.142.1 dev eth0 proto zebra
10.0.0.8/30 via 172.16.142.1 dev eth0 proto zebra
10.0.0.12/30 via 172.16.142.1 dev eth0 proto zebra
10.0.0.16/30 via 172.16.142.1 dev eth0 proto zebra
10.0.0.20/30 via 172.16.142.1 dev eth0 proto zebra
10.0.0.24/30 via 172.16.142.1 dev eth0 proto zebra
10.0.0.28/30 via 172.16.142.1 dev eth0 proto zebra
172.0.0.0/8 via 172.16.142.1 dev eth0 proto zebra
172.16.142.0/30 dev eth0 proto kernel scope link src 172.16.142.2
172.16.142.4/30 via 172.16.142.1 dev eth0 proto zebra
172.16.143.0/30 via 172.16.142.1 dev eth0 proto zebra
172.16.143.4/30 via 172.16.142.1 dev eth0 proto zebra
192.168.0.128/29 dev eth1 proto kernel scope link src 192.168.0.129
192.168.0.136/29 via 172.16.142.1 dev eth0 proto zebra
192.168.0.144/29 via 172.16.142.1 dev eth0 proto zebra
192.168.0.152/29 via 172.16.142.1 dev eth0 proto zebra
192.168.0.192/29 dev eth1 proto kernel scope link src 192.168.0.193
192.168.0.200/29 via 172.16.142.1 dev eth0 proto zebra metric 3
192.168.0.208/29 via 172.16.142.1 dev eth0 proto zebra metric 3
192.168.0.216/29 via 172.16.142.1 dev eth0 proto zebra metric 3
192.168.0.232/29 via 172.16.142.1 dev eth0 proto zebra
192.168.0.240/29 via 172.16.142.1 dev eth0 proto zebra
192.168.0.248/29 via 172.16.142.1 dev eth0 proto zebra
root@RAGaliza:/tmp/pycore.35097/RAGaliza.conf# S
```

Figura 29: Teresa não consegue responder a AfonsoHenriques

2.2.6 Alinea D II

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@Teresa:/tmp/pycore.35097/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) 0.032 ms 0.002 ms 0.003 ms
 2 172.16.142.1 (172.16.142.1) 0.016 ms 0.003 ms 0.002 ms
 3 10.0.0.2 (10.0.0.2) 0.015 ms 0.003 ms 0.003 ms
 4 10.0.0.6 (10.0.0.6) 0.016 ms 0.004 ms 0.005 ms
 5 10.0.0.18 (10.0.0.18) 0.014 ms 0.005 ms 0.005 ms
 6 10.0.0.14 (10.0.0.14) 0.025 ms 0.027 ms 0.006 ms
 7 10.0.0.26 (10.0.0.26) 0.017 ms 0.008 ms 0.007 ms
 8 10.0.0.30 (10.0.0.30) 0.017 ms 0.008 ms 0.008 ms
 9 172.16.143.2 (172.16.143.2) 0.018 ms 0.009 ms 0.008 ms
10 192.168.0.226 (192.168.0.226) 0.016 ms 0.010 ms 0.009 ms
root@Teresa:/tmp/pycore.35097/Teresa.conf#
```

Figura 30: Rota efetuada de Teresa para AfonsoHenriques

Não exatamente, no sentido AfonsoHenriques → Teresa a rota passa pelo router n1. No entanto, no sentido inverso, a rota passa pelo router n4.

2.2.7 Alinea E

- e. 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
```

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.

Essa rota não é utilizada porque tanto para CDN como para Galiza os pacotes são encaminhados por 10.0.0.25, como é possível visualizar nestes dois traceroutes:

```
root@n5:/tmp/pycore.35097/n5.conf# traceroute -I 192.168.0.128
traceroute to 192.168.0.128 (192.168.0.128), 30 hops max, 60 byte packets
 1 10.0.0.25 (10.0.0.25) 0.041 ms 0.003 ms 0.002 ms
 2 10.0.0.13 (10.0.0.13) 0.018 ms 0.003 ms 0.002 ms
 3 10.0.0.17 (10.0.0.17) 0.022 ms 0.003 ms 0.004 ms
 4 10.0.0.5 (10.0.0.5) 0.020 ms 0.005 ms 0.005 ms
 5 10.0.0.1 (10.0.0.1) 0.015 ms 0.006 ms 0.006 ms
 6 * * *
 7 * * *
 8 * * *
 9 * * *
```

Figura 31: TraceRoute para Galiza

```
root@n5:/tmp/pycore.35097/n5.conf# traceroute -I 192.168.0.136
traceroute to 192.168.0.136 (192.168.0.136), 30 hops max, 60 byte packets
 1 10.0.0.25 (10.0.0.25) 0.032 ms 0.003 ms 0.002 ms
 2 10.0.0.21 (10.0.0.21) 0.016 ms 0.003 ms 0.003 ms
 3 10.0.0.17 (10.0.0.17) 0.015 ms 0.005 ms 0.004 ms
 4 10.0.0.5 (10.0.0.5) 0.015 ms 0.005 ms 0.005 ms
 5 10.0.0.1 (10.0.0.1) 0.013 ms 0.006 ms 0.006 ms
 6 * * *
 7 * * *
 8 * * *
```

Figura 32: TraceRoute para CDN

2.2.8 Alinea F

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

Os endereços utilizados pelos 4 polos são privados de Classe C já que estão entre 192.168.0.0 e 192.168.255.255. Os endereços utilizados no core da rede/ISPs são privados de Classe A já que estão entre 10.0.0.0 to 10.255.255.255.

2.2.9 Alinea G

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

Os switches não têm IP atribuído.

Os switches são dispositivos de segunda camada que fazem a distribuição de pacotes através dos endereços MAC de cada equipamento. Ao contrário dos routers, que encaminham os pacotes usando endereços IP, os switches não necessitam de um endereço IP para funcionarem. A sua função é simplesmente interligar dispositivos dentro da mesma rede local, usando a sua tabela de comutação para saber para onde devem enviar cada pacote.

2.3 Exercício 3

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

2.3.1 Alinea A

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

Começando por remover as conexões para Galiza e para CDN:

```
root@n6:/tmp/pycore.35097/n6.conf# ip route del 192.168.0.128/29
root@n6:/tmp/pycore.35097/n6.conf# ip route del 192.168.0.136/29
root@n6:/tmp/pycore.35097/n6.conf#
```

Figura 33: Remover rotas para Galiza e CDN

Pegando nas informações sobre os routers :

Galiza : 192.168.0.128 11000000.10101000.00000000.10000000

Tendo como sub-redes:

- CDN : 192.168.0.136 11000000.10101000.00000000.10001000
- 192.168.0.144 11000000.10101000.00000000.10010000
- 192.168.0.152 11000000.10101000.00000000.10011000

Podemos observar que partilham até : 11000000.10101000.00000000.100 — 0000

Então, concluímos que o ip da supernet será: 11000000.10101000.00000000.10000000 192.168.0.128
E com Mask: 1111111.1111111.1111111.11110000 255.255.255.240 192.168.0.128/28

Podemos agora adicionar e testar a conexão :

```
root@n6:/tmp/pycore.35097/n6.conf# ip route add 192.168.0.128/28 via 10.0.0.1
root@n6:/tmp/pycore.35097/n6.conf# ping 192.168.0.137
PING 192.168.0.137 (192.168.0.137) 56(84) bytes of data:
64 bytes from 192.168.0.137: icmp_seq=1 ttl=63 time=0.094 ms
64 bytes from 192.168.0.137: icmp_seq=2 ttl=63 time=0.035 ms
64 bytes from 192.168.0.137: icmp_seq=3 ttl=63 time=0.054 ms
64 bytes from 192.168.0.137: icmp_seq=4 ttl=63 time=0.052 ms
64 bytes from 192.168.0.137: icmp_seq=5 ttl=63 time=0.051 ms
^C
--- 192.168.0.137 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4085ms
rtt min/avg/max/mdev = 0.035/0.057/0.094/0.019 ms
root@n6:/tmp/pycore.35097/n6.conf# ping 192.168.0.129
PING 192.168.0.129 (192.168.0.129) 56(84) bytes of data:
64 bytes from 192.168.0.129: icmp_seq=1 ttl=63 time=0.076 ms
64 bytes from 192.168.0.129: icmp_seq=2 ttl=63 time=0.053 ms
64 bytes from 192.168.0.129: icmp_seq=3 ttl=63 time=0.061 ms
64 bytes from 192.168.0.129: icmp_seq=4 ttl=63 time=0.043 ms
^C
```

Figura 34: Adicionar rota e executar um ping

2.3.2 Alinea B

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

```
root@n6:/tmp/pycore.35097/n6.conf# ip route del 192.168.0.232/29
root@n6:/tmp/pycore.35097/n6.conf# ip route del 192.168.0.224/29
root@n6:/tmp/pycore.35097/n6.conf# ping 192.168.0.125
ping: connect: Network is unreachable
root@n6:/tmp/pycore.35097/n6.conf#
```

Figura 35: Remover rotas para CondadoPortucalense e Institucional

Pegando nas informações sobre os routers :

RACondado : 192.168.0.224 11000000.10101000.00000000.11100000

Tendo como sub-redes:

- RAIstitucional : 192.168.0.232 11000000.10101000.00000000.11101000
- 192.168.0.240 11000000.10101000.00000000.11110000
- 192.168.0.248 11000000.10101000.00000000.11111000

Podemos observar que partilham até : 11000000.10101000.00000000.1111 — 00000

Então, concluímos que o ip da supernet será: 11000000.10101000.00000000.11100000 192.168.0.224
E com Mask: 1111111.1111111.1111111.11100000 255.255.255.224 192.168.0.128/27

Podemos agora adicionar e testar a conexão :

```
root@n6:/tmp/pycore.35097/n6.conf# ip route add 192.168.0.224/27 via 10.0.0.6
```

Figura 36: Adição de rotas supernet para CondadoPortucalense e Institucional

```
root@n6:/tmp/pycore.35097/n6.conf# ping 192.168.0.225
PING 192.168.0.225 (192.168.0.225) 56(84) bytes of data:
64 bytes from 192.168.0.225: icmp_seq=1 ttl=59 time=0.182 ms
64 bytes from 192.168.0.225: icmp_seq=2 ttl=59 time=0.085 ms
64 bytes from 192.168.0.225: icmp_seq=3 ttl=59 time=0.086 ms
64 bytes from 192.168.0.225: icmp_seq=4 ttl=59 time=0.088 ms
64 bytes from 192.168.0.225: icmp_seq=5 ttl=59 time=0.088 ms
^C
--- 192.168.0.225 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4075ms
rtt min/avg/max/mdev = 0.085/0.105/0.182/0.038 ms
root@n6:/tmp/pycore.35097/n6.conf# ping 192.168.241
PING 192.168.241 (192.168.0.241) 56(84) bytes of data:
64 bytes from 192.168.0.241: icmp_seq=1 ttl=59 time=0.156 ms
64 bytes from 192.168.0.241: icmp_seq=2 ttl=59 time=0.118 ms
64 bytes from 192.168.0.241: icmp_seq=3 ttl=59 time=0.081 ms
64 bytes from 192.168.0.241: icmp_seq=4 ttl=59 time=0.100 ms
```

Figura 37: Adição de rotas supernet para CondadoPortucalense e Institucional

2.3.3 Alinea C

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

Aspectos Positivos do Supernetting (Agregação de Rotas):

- Redução da Tabela de Encaminhamento – Diminui o número de entradas, poupando recursos (CPU/memória) nos routers.
- Gestão Simplificada – Facilita a configuração e manutenção de redes de grande escala.
- Melhor Desempenho – Reduz o processamento desnecessário nos dispositivos de rede.

Aspectos Negativos do Supernetting:

- Perda de Controlo Granular – Sub-redes perdem políticas individuais (ex.: QoS, ACLs).
- Risco de Agregação Indesejada – Se mal planeado, pode incluir redes não relacionadas.
- Dificuldade em Diagnosticar Problemas – Complexifica a identificação de falhas em sub-redes específicas.

Resumo: Ótimo para escalabilidade, mas requer planeamento rigoroso para evitar efeitos indesejados.