



UNIVERSIDADE DO MINHO

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Redes de Computadores - RC  
Trabalho Prático 2

Bernardo Saraiva (A93189)  
José Gonçalves (A93204)  
Gonçalo Santos (A93279)

25/03/2022

# Conteúdo

<b>1</b>	<b>Questões e Respostas</b>	<b>4</b>
1.1	Questão 1 . . . . .	4
1.1.1	Alínea a . . . . .	4
1.1.2	Alínea b . . . . .	5
1.1.3	Alínea c . . . . .	6
1.1.4	Alínea d . . . . .	7
1.1.5	Alínea e . . . . .	8
1.2	Questão 2 . . . . .	8
1.2.1	Alínea a . . . . .	9
1.2.2	Alínea b . . . . .	9
1.2.3	Alínea c . . . . .	9
1.2.4	Alínea d . . . . .	10
1.2.5	Alínea e . . . . .	10
1.2.6	Alínea f . . . . .	11
1.2.7	Alínea g . . . . .	12
1.3	Questão 3 . . . . .	12
1.3.1	Alínea a . . . . .	12
1.3.2	Alínea b . . . . .	12
1.3.3	Alínea c . . . . .	13
1.3.4	Alínea d . . . . .	13
1.3.5	Alínea e . . . . .	13
1.3.6	Alínea f . . . . .	13
1.3.7	Alínea g . . . . .	14
<b>2</b>	<b>PARTE II</b>	<b>15</b>
2.1	Exercício 1 . . . . .	15
2.1.1	Alínea a . . . . .	15
2.1.2	Alínea b . . . . .	16
2.1.3	Alínea c . . . . .	16
2.1.4	Alínea d . . . . .	16
2.1.5	Alínea e . . . . .	17
2.1.6	Alínea f . . . . .	18

2.2	Exercício 2 . . . . .	19
2.2.1	Alínea a . . . . .	19
2.2.2	Alínea b . . . . .	21
2.2.3	Alínea c . . . . .	22
2.2.4	Alínea d . . . . .	22
2.2.5	Alínea e . . . . .	22
2.3	Exercício 3 . . . . .	23
2.3.1	Alínea 1 . . . . .	23
2.3.2	Alínea 2 . . . . .	26
2.3.3	Alínea 3 . . . . .	26
<b>3</b>	<b>Conclusão</b>	<b>29</b>

# Capítulo 1

## Questões e Respostas

### 1.1 Questão 1

1. Prepare uma topologia CORE para verificar o comportamento do traceroute. Na topologia deve existir: um host (pc) cliente designado Bela cujo router de acesso é R2; o router R2 está simultaneamente ligado a dois routers R3 e R4; estes estão conectados a um router R5, que por sua vez, se liga a um host (servidor) designado Monstro. Ajuste o nome dos equipamentos atribuídos por defeito para o enunciado. Nas ligações (links) da rede de core estabeleça um tempo de propagação de 10ms. Após ativar a topologia, note que pode não existir conectividade IP imediata entre a Bela e o Monstro até que o anúncio de rotas entre routers estabilize.

#### 1.1.1 Alínea a

Active o wireshark ou o tcpdump no host Bela. Numa shell de Bela execute o comando `traceroute -I` para o endereço IP do Monstro.

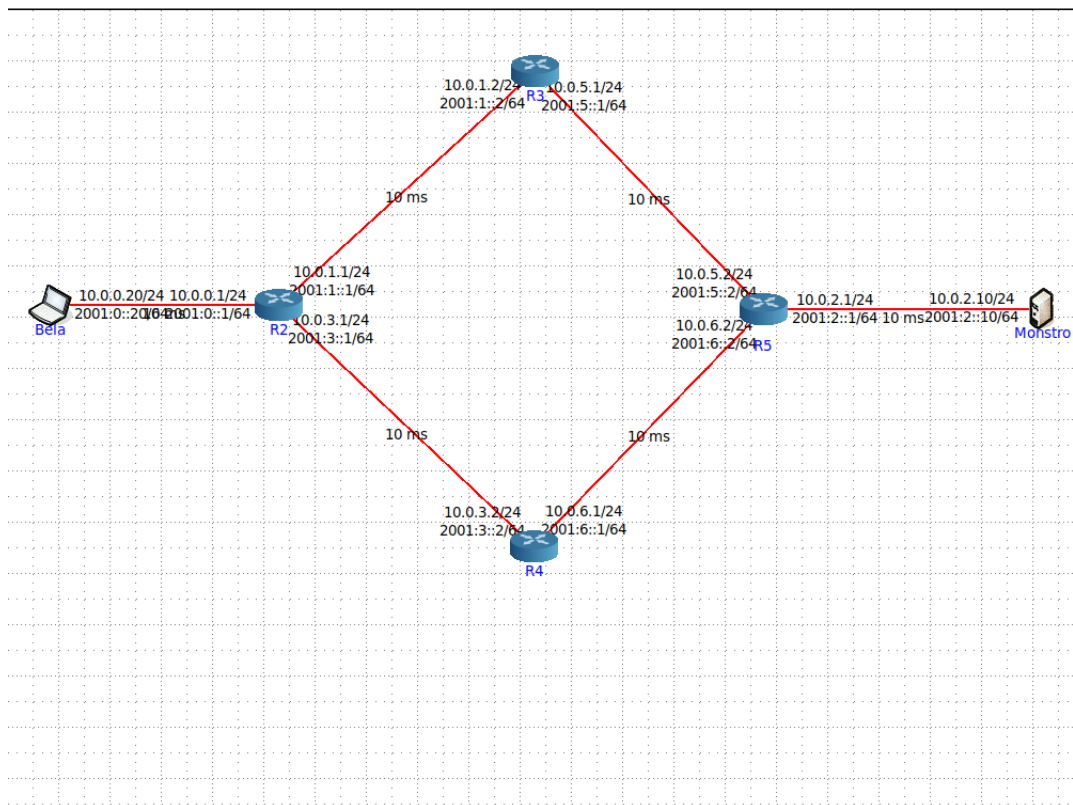


Figura 1.1: Topologia criada com base no enunciado

```

vcmd
root@Bela:/tmp/pycore_42909/Bela.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) 20.367 ms 20.323 ms 20.313 ms
 2 10.0.1.2 (10.0.1.2) 40.548 ms 40.541 ms 40.532 ms
 3 10.0.3.2 (10.0.3.2) 60.673 ms 60.666 ms 60.659 ms
 4 10.0.5.10 (10.0.5.10) 80.867 ms 80.860 ms 80.851 ms
root@Bela:/tmp/pycore_42909/Bela.conf#

```

Figura 1.2: Execução do comando traceroute -I na shell de Bela para o Monstro

### 1.1.2 Alínea b

Registe e analise o tráfego ICMP enviado pelo sistema Bela e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

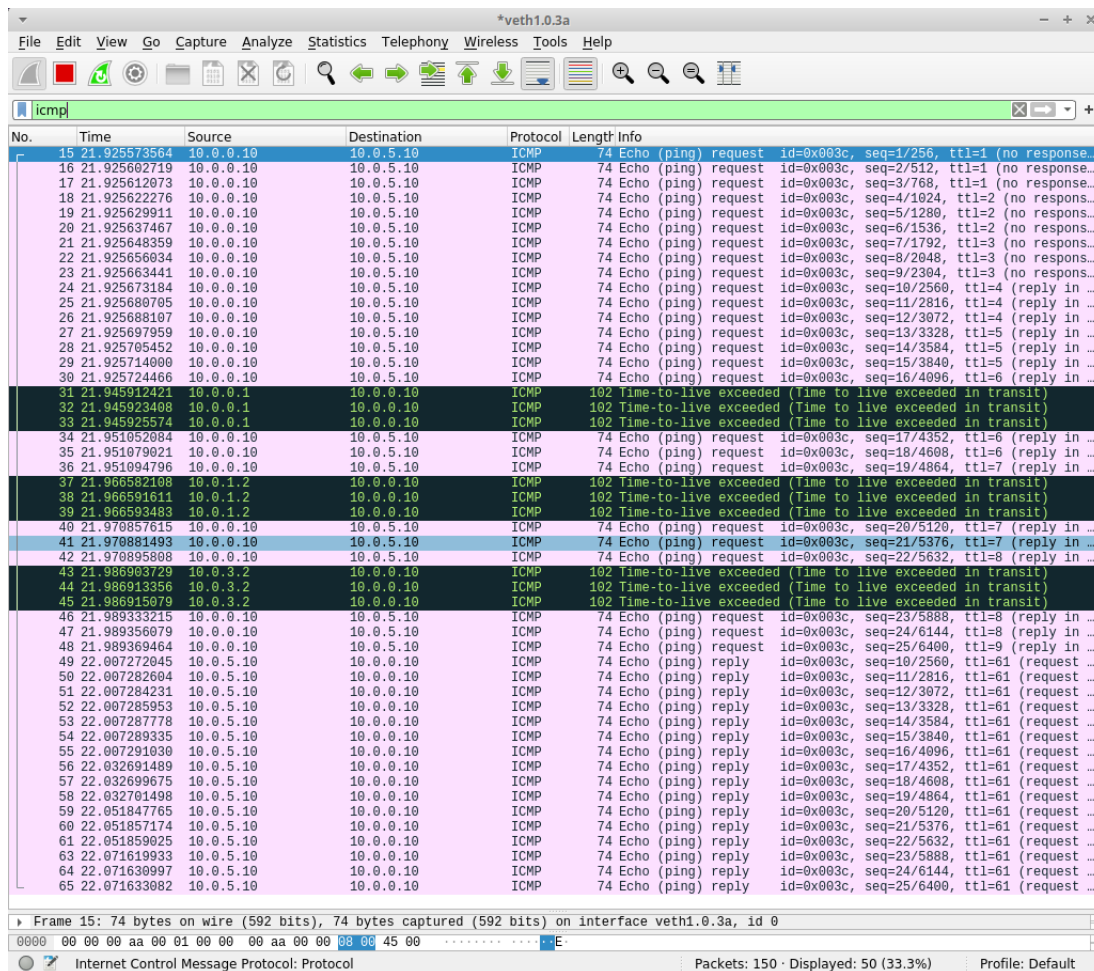


Figura 1.3: Tráfego ICMP entre o host Bela e Monstro

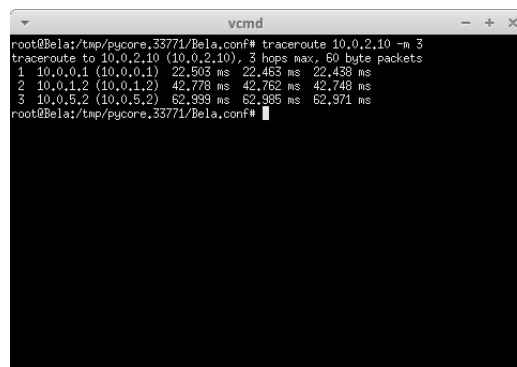
Através do traceroute, é possível mapear a rede desde uma origem a um destino com este propósito são enviados pacotes ICMP através da rede que passam por todos os seus nodos de forma a obter a sua informação. Para este efeito, e como é possível verificar na Figura 1.3 são enviados pacotes ICMP com TTL's(time to live) sucessivamente maiores, até que seja obtida uma resposta e neste momento param de ser enviados pacotes, dado que o destino já foi alcançado com um TTL inferior ao destes. No que diz respeito aos pacotes que relatam a insuficiência de TTL estes possuem como source 3 dos diferentes routers da rede o que comprova exatamente que os pacotes são reencaminhados por toda a topologia à exceção do R4, que não oferece um caminho suficientemente bom para ser escolhido.

### 1.1.3 Alínea c

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

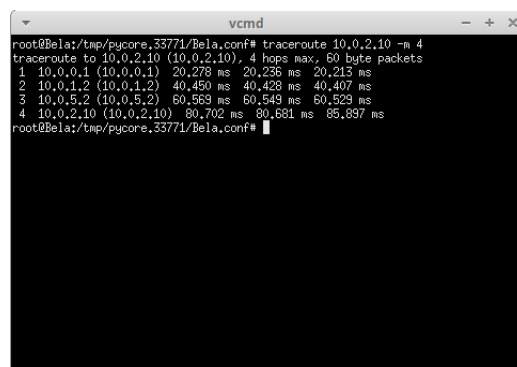
Como é possível entender através da captura efetuada são necessários 4 saltos entre o Bela e o Monstro, assim como seria expectável ao analisar a topologia da rede encontrada na Figura 1.1, podemos inferir este resultado dado que são recebidos 3 blocos de pacotes a informar que o TTL foi excedido, o que prova que o TTL mínimo necessário para alcançar o Monstro através de Bela é 4.

Para verificar na prática a assunção a cima, é executado o comando presente na Figura 1.4 e Figura 1.5. Como verificado, ao executar o comando traceroute com TTL de 3, não é alcançado o endereço 10.0.2.10 identificador do host monstro, já com TTL 4 é possível alcançar o mesmo, comprovando desta forma a premissa.



```
vcmd
root@Bela:/tmp/pycore,33771/Bela.conf# traceroute 10.0.2.10 -m 3
traceroute to 10.0.2.10 (10.0.2.10), 3 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  22.503 ms  22.463 ms  22.438 ms
 2  10.0.1.2 (10.0.1.2)  42.778 ms  42.762 ms  42.748 ms
 3  10.0.5.2 (10.0.5.2)  62.999 ms  62.388 ms  62.971 ms
root@Bela:/tmp/pycore,33771/Bela.conf#
```

Figura 1.4: Comando traceroute com ttl max de 3

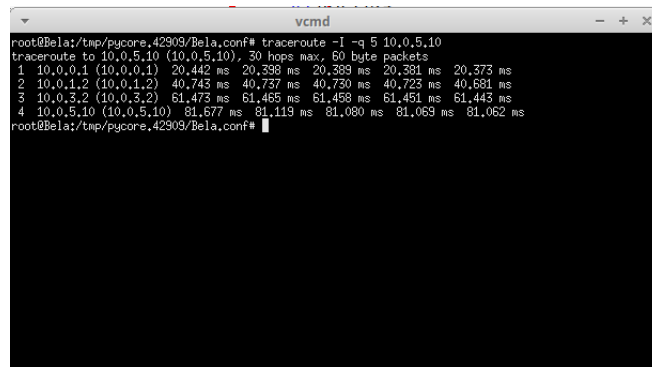


```
vcmd
root@Bela:/tmp/pycore,33771/Bela.conf# traceroute 10.0.2.10 -m 4
traceroute to 10.0.2.10 (10.0.2.10), 4 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  20.278 ms  20.236 ms  20.215 ms
 2  10.0.1.2 (10.0.1.2)  40.460 ms  40.428 ms  40.407 ms
 3  10.0.5.2 (10.0.5.2)  60.569 ms  60.543 ms  60.529 ms
 4  10.0.2.10 (10.0.2.10)  80.702 ms  80.681 ms  85.897 ms
root@Bela:/tmp/pycore,33771/Bela.conf#
```

Figura 1.5: Comando traceroute com ttl max de 4

#### 1.1.4 Alínea d

Calcule o valor médio do tempo de ida-e-volta (RTT - Round-Trip Time) obtido no acesso ao servidor. Para melhorar a média, poderá alterar o número pacotes de prova com a opção -q.



```
root@Bela:/tmp/pycore.42909/Bela.conf# traceroute -I -q 5 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) 20.442 ms 20.539 ms 20.563 ms 20.581 ms 20.373 ms
 2 10.0.1.2 (10.0.1.2) 40.745 ms 40.737 ms 40.730 ms 40.723 ms 40.681 ms
 3 10.0.3.2 (10.0.3.2) 61.473 ms 61.465 ms 61.458 ms 61.451 ms 61.443 ms
 4 10.0.5.10 (10.0.5.10) 81.677 ms 81.119 ms 81.080 ms 81.069 ms 81.062 ms
root@Bela:/tmp/pycore.42909/Bela.conf#
```

Figura 1.6: Comando auxiliar para o cálculo do RTT

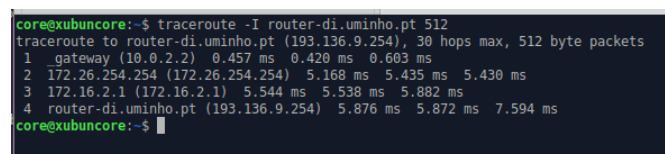
Como é possível verificar na Figura 1.6, para o servidor Monstro (10.0.5.10) a partir de Bela os tempos de ida-e-volta para os 5 pacotes enviados foram de, respetivamente: 81.677 ms, 81.119 ms, 81.080 ms, 81.069 ms, 81.062 ms. Como tal, o RTT médio seria de aproximadamente 81.2014.

### 1.1.5 Alínea e

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?

Ao dividir o RTT por dois, seria calculado uma estimativa do One-Way Delay. No entanto este valor poder-se-ia encontrar bastante longe da realidade, já que nada garante que o percurso selecionado para o envio do pacote de request seja o mesmo do pacote de reply. Desta forma, torna-se difícil prever qual o One-Way Delay através deste método.

## 1.2 Questão 2



```
core@xubuncore:~$ traceroute -I router-di.uminho.pt 512
traceroute to router-di.uminho.pt (193.136.9.254), 30 hops max, 512 byte packets
 1 gateway (10.0.2.2) 0.457 ms 0.420 ms 0.603 ms
 2 172.26.254.254 (172.26.254.254) 5.168 ms 5.435 ms 5.430 ms
 3 172.16.2.1 (172.16.2.1) 5.544 ms 5.538 ms 5.882 ms
 4 router-di.uminho.pt (193.136.9.254) 5.876 ms 5.872 ms 7.594 ms
core@xubuncore:~$
```

Figura 1.7: Output do traceroute para *router-di.uminho.pt* com tamanho do pacote 512

O *traceroute* demonstrado anteriormente pode ser explicado uma vez que inicialmente vai ao router que está na sala de aula e este redirecciona-o para outro. Desse outro, para um 3º e finalmente chega ao DI. Neste caso identifica-se o tamanho do pacote como 512 bytes



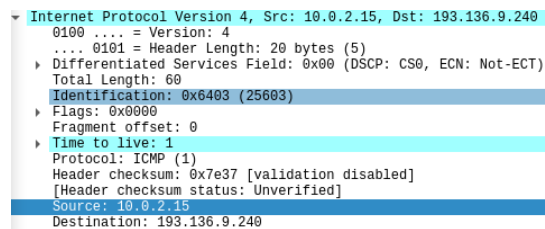
Captura do tráfego do traceroute: i) traceroute -I marco.uminho.pt

```
core@xubuncore:~$ traceroute -I marco.uminho.pt
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 60 byte packets
 1  gateway (10.0.2.2)  0.699 ms  0.668 ms  0.657 ms
 2  172.26.254.254 (172.26.254.254)  5.250 ms  5.241 ms  5.231 ms
 3  172.16.2.1 (172.16.2.1)  5.218 ms  5.154 ms  5.140 ms
 4  172.16.115.252 (172.16.115.252)  5.129 ms  5.960 ms  5.951 ms
 5  marco.uminho.pt (193.136.9.240)  5.941 ms  5.932 ms  5.920 ms
core@xubuncore:~$
```

Figura 1.8: Output do traceroute para *router-di.uminho.pt* sem tamanho de pacote definido

### 1.2.1 Alínea a

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



```
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.136.9.240
 0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0x6403 (25603)
  Flags: 0x0000
  Fragment offset: 0
  Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x7e37 [validation disabled]
  [Header checksum status: Unverified]
  Source: 10.0.2.15
  Destination: 193.136.9.240
```

Figura 1.9: Informação da captura

O endereço IP da interface activa é o 10.0.2.15. Podemos afirmar esta conclusão ao verificar que o campo *Source* apresenta este valor.

### 1.2.2 Alínea b

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

O valor do campo protocolo é ICMP(1), sendo que o valor 1 referencia o protocolo ICMP. O ICMP que aparece antes é uma ajuda do Wireshark para se saber imediatamente que se está perante ICMP.

### 1.2.3 Alínea 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?

- Cabeçalho = 20 bytes
- Total Length = 60 bytes

- Total Length = Cabeçalho + Payload
- $\Leftrightarrow \text{Payload} = \text{TotalLength} - \text{Cabeçalho}$   
 $\Leftrightarrow \text{Payload} = 60 - 20$   
 $\Leftrightarrow \text{Payload} = 40 \text{ bytes}$

#### 1.2.4 Alínea d

d. O datagrama IP foi fragmentado? Justifique.

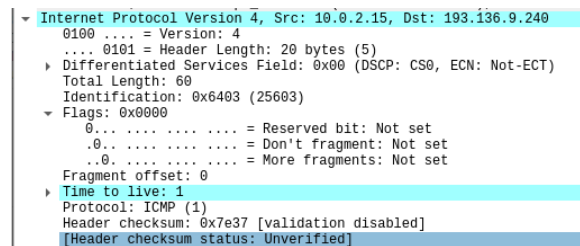


Figura 1.10: Informação relativa a um dos pacotes capturados

Através das flags é possível verificar que se tem o campo *more fragments* a zero. Como o offset é zero também, confirma-se que o datagrama IP não foi fragmentado.

#### 1.2.5 Alínea e

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

No.	Time	Source	Destination	Protocol	Length	Info
5	0.006148221	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1256, ttl=64
6	0.00617251	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1257, ttl=64
7	0.00617885	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1258, ttl=64
8	0.00618708	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1259, ttl=64
9	0.006198134	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1260, ttl=64
10	0.006208182	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1261, ttl=64
11	0.006220785	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1262, ttl=64
12	0.006235989	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1263, ttl=64
13	0.006250966	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1264, ttl=64
14	0.006309961	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1265, ttl=64
15	0.006319275	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1266, ttl=64
16	0.006328566	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1267, ttl=64
17	0.006338882	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1268, ttl=64
18	0.006348887	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1269, ttl=64
19	0.006360275	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1270, ttl=64
20	0.006440216	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1271, ttl=64
41	0.013026511	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1485, ttl=64
42	0.013053789	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1486, ttl=64
43	0.013066555	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1487, ttl=64
21	0.006327359	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1272, ttl=64
22	0.006337229	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1273, ttl=64
23	0.006342294	10.0.2.15	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1274, ttl=64
33	0.011430384	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1275, ttl=64
37	0.012277509	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1276, ttl=64
39	0.012277603	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1277, ttl=64
27	0.011430622	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1278, ttl=64
30	0.011430624	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1279, ttl=64
32	0.011430641	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1280, ttl=64
28	0.011430619	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1281, ttl=64
29	0.011430619	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1282, ttl=64
31	0.011430607	172.16.115.252	193.136.9.240	ICMP	74	Echo (ping) request 10.0.2.15 seq=1283, ttl=64

Figura 1.11: Comparação entre mensagens ordenadas pelo seu campo source para análise dos campos que variam

Com base na captura disponível na imagem anterior, é possível verificar que variam os valores correspondentes ao *Time To Live (TTL)*, *Identification* e *Header Checksum*.

No caso atual, é possível reparar que o *offset* se mantém sempre a zero (não varia), pelo que se pode afirmar que não ocorre fragmentação, provando que a mensagem não necessita de ser particionada para o envio.

## 1.2.6 Alínea f

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

Após ordenação, é possível verificar que o campo de Identificação do datagrama IP aumenta gradualmente (de 1 em 1) após análise da sequência de tráfego ICMP gerado a partir do endereço IP atribuído à interface da sua máquina.

O TTL, se for visto por ordem de endereço fonte tal como referido anteriormente, mantém-se constante ou aumenta. Deste modo, é possível perceber que a cada falha na entrega do pacote ao seu destino, o valor deste campo incrementa de maneira a analisar se é possível entregar o datagrama para o nodo seguinte.

### 1.2.7 Alínea g

g. Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP TTL exceeded enviadas ao seu computador. Qual é o valor do campo TTL? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL exceeded enviados ao seu host? Porquê?

Quando uma mensagem não consegue alcançar o destino pretendido, o router que recebe a mensagem por último envia a resposta de volta à origem com TTL 256. A cada salto que faça para voltar ao nodo de origem, é decrementado um salto nesse valor de TTL.

Neste caso, podemos reparar que o valor do campo TTL nas respostas ICMP TTL exceeded enviadas ao computador varia entre 253 e 255, pelo que se conclui que o último nodo antes de alcançar o destino pretendido se encontra a 3 saltos do router original.

## 1.3 Questão 3

Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempo de captura. Observe o tráfego depois do tamanho de pacote ter sido definido para  $(4000 + X)$  bytes.

### 1.3.1 Alínea a

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

Após localizar a primeira mensagem de ICMP, é possível concluir que existiu a necessidade de o pacote ser fragmentado visto que este ultrapassou o MTU definido.

### 1.3.2 Alínea 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 protocolo de IPv4 contém a flag *More fragments*, que caso esteja activa indica que existe mais fragmentos deste pacote, ou seja que o pacote esta incompleto. Neste pacote em específico é possível observar o seu campo *offset* corresponde a 0, o que indica, que este fragmento contém a informação inicial do seu pacote, sendo

que o datagrama na sua totalidade é constituído por 1514 bytes, mas a componente do IP ocupara 1500 bytes.

### 1.3.3 Alínea c

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

Como referido na questão anterior e possível descobrir que ainda existem mais fragmentos deste pacote, uma vez que, a flag *More Fragments* continua activa, também é possível descobrir que não é o primeiro fragmento do pacote visto que o *offset* do fragmento é diferente de zero .

### 1.3.4 Alínea d

**d. Quantos fragmentos foram criados a partir do datagrama original?**

Foi necessária a criação de 3 fragmentos a partir do datagrama original.

### 1.3.5 Alínea 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 diferem no cabeçalho IP entre os diferentes fragmentos são a flag, *More Fragments* e o *offset*. O campo *More Fragments* permite descobrir que o pacote se encontra fragmentado e que existe mais fragmentos do datagrama, e o campo *offset* indica o seu offset em relação à origem do seu pacote original, fornecendo assim a sua posição. Estes dois componentes do cabeçalho permitem assim ao receptor da informação reconstruir o datagrama original.

### 1.3.6 Alínea f

**f. Verifique o processo de fragmentação através de um processo de cálculo.**

**Fragmento 1:**

$L = 1514$  bytes

Cabeçalho Ethernet = 14 bytes

Cabeçalho IP = 20 bytes

Offset = 0 bytes

More Fragments = 1

$\text{dados} = L - \text{Cabeçalho Ethernet} - \text{Cabeçalho IP} = 1514 - 14 - 20 = 1480 \text{ bytes}$

**Fragmento 2:**

$L = 1514 \text{ bytes}$

Cabeçalho Ethernet = 14 bytes

Cabeçalho IP = 20 bytes

offset = 1480 bytes

More Fragments = 1 bytes

$\text{dados} = L - \text{Cabeçalho Ethernet} - \text{Cabeçalho IP} = 1514 - 14 - 20 = 1480 \text{ bytes}$

$\text{dados recebidos} = 1480 + 1480 = 2960 \text{ bytes}$

**Fragmento 3:**

$L = 1514 \text{ bytes}$

Cabeçalho Ethernet = 14 bytes

Cabeçalho IP = 20 bytes

More Fragments = 0

$\text{dados} = L - \text{Cabeçalho Ethernet} - \text{Cabeçalho IP} = 1205 - 14 - 20 = 1171 \text{ bytes}$

$\text{dados recebidos: } 2960 + 1171 = 4131 \text{ bytes}$

### 1.3.7 Alínea g

**g. Escreva uma expressão lógica que permita detectar o último fragmento correspondente ao datagrama original.**

Se  $\text{soma}(\text{tamanho fragmento anteriores}) == \text{offset}(\text{fragmento})$  and  $\text{MoreFragments}(\text{fragmento}) == 0$  and  $\text{tamanho}(\text{fragmento}) \leq MTU$  and  $\text{id}(\text{fragmentos anteriores}) == \text{id}(\text{fragmento})$

## Capítulo 2

# PARTE II

### 2.1 Exercício 1

#### 2.1.1 Alínea a

a. Indique que endereços IP e máscaras de rede foram atribuídos pelo CORE a cada equipamento. Para simplificar, pode incluir uma imagem que ilustre de forma clara a topologia definida e o endereçamento usado.

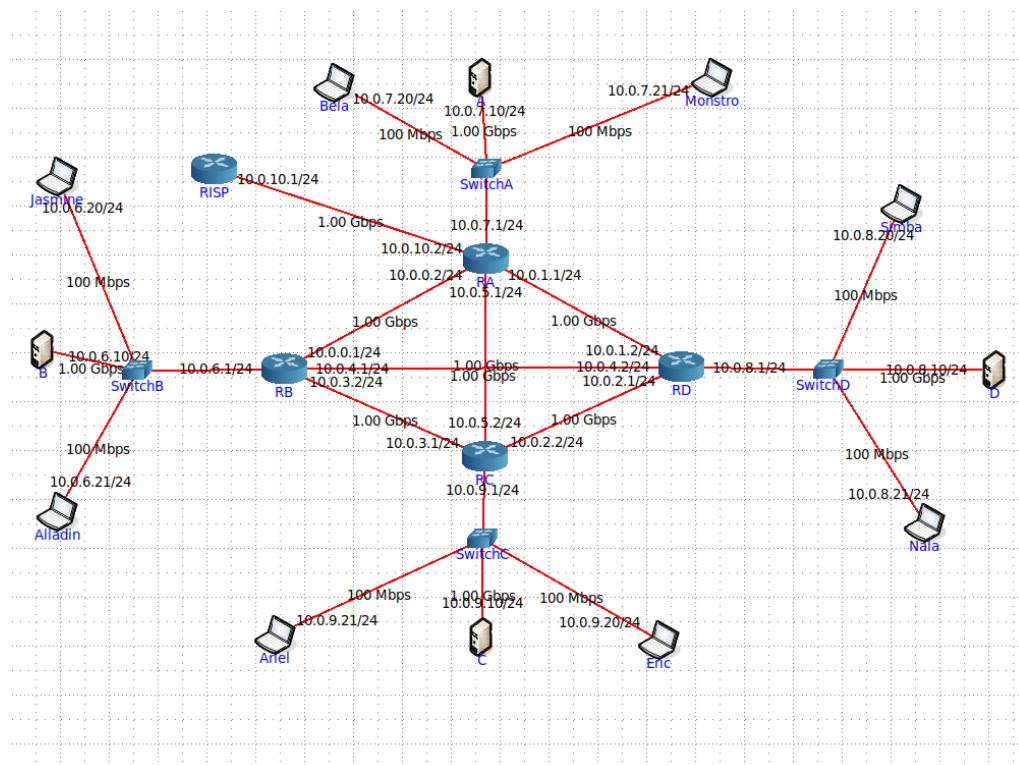


Figura 2.1: Topologia desenvolvida

Jasmine: 10.0.6.20 Alladin:10.0.6.21

Bela: 10.0.7.20 Monstro: 10.0.7.21

Ariel: 10.0.9.21 Eric: 10.0.9.20

Simba: 10.0.8.20 Nala: 10.0.8.21

Relativamente à máscara de rede, esta é igual a 24 para todos.

### **2.1.2 Alínea b**

#### **b. Tratam-se de endereços públicos ou privados? Porquê?**

A atribuição de endereços IP pode ser feita com endereços públicos ou privados, sendo que os endereços privados podem estar contidos em uma das seguintes 3 gamas:

- Class A: 10.0.0.0 — 10.255.255.255
- Class B: 172.16.0.0 — 172.31.255.255
- Class C: 192.168.0.0 — 192.168.255.255

Após analisar a gama de IP's atribuída pelo CORE aos equipamentos e as classes descritas acima, é possível concluir que todos os IP's atribuídos são privados, uma vez que se inserem na Class A.

### **2.1.3 Alínea c**

#### **c. Porque razão não é atribuído um endereço IP aos switches?**

O facto de os switches não terem atribuído um endereço IP deve-se ao facto de estes serem de nível 2 no modelo OSI, sendo que não reconhecem a tecnologia IP.

### **2.1.4 Alínea d**

#### **d. Usando o comando ping certifique-se que existe conectividade IP interna a cada departamento (e.g. entre um laptop e o servidor respetivo).**

Para verificar a conectividade IP interna de cada departamento, recorreu-se ao comando ping, abrindo o terminal de um dos laptops e fazendo ping ao endereço do servidor respetivo. Nas imagens seguintes demonstram-se os resultados obtidos.



```

root@Bela:/tmp/pycore.41465/Bela.conf# ping 10.0.7.10
PING 10.0.7.10 (10.0.7.10) 56(84) bytes of data.
64 bytes from 10.0.7.10: icmp_seq=1 ttl=64 time=0.822 ms
64 bytes from 10.0.7.10: icmp_seq=2 ttl=64 time=0.361 ms
64 bytes from 10.0.7.10: icmp_seq=3 ttl=64 time=0.254 ms
^C
--- 10.0.7.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2025ms
rtt min/avg/max/mdev = 0.254/0.479/0.822/0.246 ms

root@Alladin:/tmp/pycore.41465/Alladin.conf# ping 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=64 time=0.757 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=64 time=0.413 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=64 time=0.271 ms
^C
--- 10.0.6.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.271/0.480/0.757/0.204 ms

root@Eric:/tmp/pycore.41465/Eric.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_seq=1 ttl=64 time=0.844 ms
64 bytes from 10.0.9.10: icmp_seq=2 ttl=64 time=0.256 ms
64 bytes from 10.0.9.10: icmp_seq=3 ttl=64 time=0.272 ms
^C
--- 10.0.9.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.256/0.457/0.844/0.273 ms

root@Nala:/tmp/pycore.41465/Nala.conf# ping 10.0.8.1
PING 10.0.8.1 (10.0.8.1) 56(84) bytes of data.
64 bytes from 10.0.8.1: icmp_seq=1 ttl=64 time=0.586 ms
64 bytes from 10.0.8.1: icmp_seq=2 ttl=64 time=0.347 ms
64 bytes from 10.0.8.1: icmp_seq=3 ttl=64 time=0.377 ms
^C
--- 10.0.8.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2054ms
rtt min/avg/max/mdev = 0.347/0.436/0.586/0.106 ms

```

Figura 2.2: Resultados do comando ping entre cada servidor e um dos seus respectivos laptops

### 2.1.5 Alínea e

e. Execute o número mínimo de comandos ping que lhe permite verificar a existência de conectividade IP entre departamentos.

De modo a garantir que se tem comunicação entre todos os departamentos, e assumindo que a configuração de todas as subredes se encontra de modo esperado, efetuou-se o comando ping entre cada par de departamentos de modo a garantir que é possível a conectividade entre ambos. Assim, assume-se que, caso exista conectividade entre um dos laptops, também existe com os restantes laptops dos departamentos em questão.

```

root@Alladin:/tmp/pycore.41465/Alladin.conf# ping 10.0.7.20
PING 10.0.7.20 (10.0.7.20) 56(84) bytes of data.
64 bytes from 10.0.7.20: icmp_seq=1 ttl=62 time=0.825 ms
64 bytes from 10.0.7.20: icmp_seq=2 ttl=62 time=0.735 ms
64 bytes from 10.0.7.20: icmp_seq=3 ttl=62 time=0.653 ms
^C
--- 10.0.7.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2033ms
rtt min/avg/max/mdev = 0.653/0.737/0.825/0.070 ms
root@Alladin:/tmp/pycore.41465/Alladin.conf#

```

Figura 2.3: Resultados do comando ping entre os departamentos A e B

```

root@Bela:/tmp/pycore.41465/Bela.conf# ping 10.0.9.21
PING 10.0.9.21 (10.0.9.21) 56(84) bytes of data.
64 bytes from 10.0.9.21: icmp_seq=1 ttl=62 time=0.828 ms
64 bytes from 10.0.9.21: icmp_seq=2 ttl=62 time=0.773 ms
64 bytes from 10.0.9.21: icmp_seq=3 ttl=62 time=0.836 ms
^C
--- 10.0.9.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.773/0.812/0.836/0.028 ms
root@Bela:/tmp/pycore.41465/Bela.conf#

```

Figura 2.4: Resultados do comando ping entre os departamentos A e C

```

root@Bela:/tmp/pycore.41465/Bela.conf# ping 10.0.8.20
PING 10.0.8.20 (10.0.8.20) 56(84) bytes of data.
64 bytes from 10.0.8.20: icmp_seq=1 ttl=62 time=1.20 ms
64 bytes from 10.0.8.20: icmp_seq=2 ttl=62 time=0.748 ms
64 bytes from 10.0.8.20: icmp_seq=3 ttl=62 time=1.04 ms
^C
--- 10.0.8.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2021ms
rtt min/avg/max/mdev = 0.748/0.996/1.199/0.187 ms
root@Bela:/tmp/pycore.41465/Bela.conf#

```

Figura 2.5: Resultados do comando ping entre os departamentos A e D

```

root@Alladin:/tmp/pycore.41465/Alladin.conf# ping 10.0.9.21
PING 10.0.9.21 (10.0.9.21) 56(84) bytes of data.
64 bytes from 10.0.9.21: icmp_seq=1 ttl=62 time=1.12 ms
64 bytes from 10.0.9.21: icmp_seq=2 ttl=62 time=0.732 ms
64 bytes from 10.0.9.21: icmp_seq=3 ttl=62 time=0.630 ms
^C
--- 10.0.9.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2011ms
rtt min/avg/max/mdev = 0.630/0.828/1.123/0.212 ms
root@Alladin:/tmp/pycore.41465/Alladin.conf#

```

Figura 2.6: Resultados do comando ping entre os departamentos B e C

```

root@Alladin:/tmp/pycore.41465/Alladin.conf# ping 10.0.8.21
PING 10.0.8.21 (10.0.8.21) 56(84) bytes of data.
64 bytes from 10.0.8.21: icmp_seq=1 ttl=62 time=0.841 ms
64 bytes from 10.0.8.21: icmp_seq=2 ttl=62 time=0.687 ms
64 bytes from 10.0.8.21: icmp_seq=3 ttl=62 time=0.650 ms
^C
--- 10.0.8.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.650/0.726/0.841/0.082 ms
root@Alladin:/tmp/pycore.41465/Alladin.conf#

```

Figura 2.7: Resultados do comando ping entre os departamentos B e D

```

root@Eric:/tmp/pycore.41465/Eric.conf# ping 10.0.8.21
PING 10.0.8.21 (10.0.8.21) 56(84) bytes of data.
64 bytes from 10.0.8.21: icmp_seq=1 ttl=62 time=0.924 ms
64 bytes from 10.0.8.21: icmp_seq=2 ttl=62 time=0.705 ms
64 bytes from 10.0.8.21: icmp_seq=3 ttl=62 time=0.999 ms
^C
--- 10.0.8.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.705/0.876/0.999/0.124 ms
root@Eric:/tmp/pycore.41465/Eric.conf#

```

Figura 2.8: Resultados do comando ping entre os departamentos C e D

### 2.1.6 Alínea f

f. Verifique se existe conectividade IP do portátil Bela para o router de acesso R ISP.

Para verificar a existência de conectividade do portátil Bela para o router de acesso R ISP, foi efetuado um ping, que indicou a existência de conectividade.

```
root@Bela:/tmp/pycore.41465/Bela.conf# ping 10.0.10.1
PING 10.0.10.1 (10.0.10.1) 56(84) bytes of data:
64 bytes from 10.0.10.1: icmp_seq=1 ttl=63 time=0.879 ms
64 bytes from 10.0.10.1: icmp_seq=2 ttl=63 time=0.485 ms
64 bytes from 10.0.10.1: icmp_seq=3 ttl=63 time=0.522 ms
^C
--- 10.0.10.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2037ms
rtt min/avg/max/mdev = 0.485/0.628/0.879/0.177 ms
```

Figura 2.9: Resultados do comando ping entre Bela e o router RISP

## 2.2 Exercício 2

### 2.2.1 Alínea a

**a. Execute o comando `netstat -rn` por forma a poder consultar a tabela de encaminhamento unicast (IPv4). Inclua no seu relatório as tabelas de encaminhamento obtidas; interprete as várias entradas de cada tabela. Se necessário, consulte o manual respetivo (`man netstat`).**

Entradas no Servidor A:

10.0.0.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, a que este router pertence

10.0.1.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, a que este router pertence

10.0.5.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, a que este router pertence

10.0.7.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, a que este router pertence

10.0.10.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, a que este router pertence

10.0.2.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, indicado que este dispositivo não lhe tem acesso direto, e que o próximo salto é para 10.0.1.2

10.0.3.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, indicado que este dispositivo não lhe tem acesso direto, e que o próximo salto é para 10.0.0.1

10.0.4.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, indicado que este dispositivo não lhe tem acesso direto, e que o próximo salto é para

10.0.0.1

10.0.6.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, indicado que este dispositivo não lhe tem acesso direto, e que o próximo salto é para 10.0.0.1

10.0.8.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, indicado que este dispositivo não lhe tem acesso direto, e que o próximo salto é para 10.0.1.2

10.0.9.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, indicado que este dispositivo não lhe tem acesso direto, e que o próximo salto é para 10.0.5.2

```
root@RA:/tmp/pycore.44587/RA.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
10.0.2.0 10.0.1.2 255.255.255.0 UG 0 0 0 eth1
10.0.3.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.4.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.5.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
10.0.6.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 0.0.0.0 255.255.255.0 U 0 0 0 eth3
10.0.8.0 10.0.1.2 255.255.255.0 UG 0 0 0 eth1
10.0.9.0 10.0.5.2 255.255.255.0 UG 0 0 0 eth2
10.0.10.0 0.0.0.0 255.255.255.0 U 0 0 0 eth4
root@RA:/tmp/pycore.44587/RA.conf#
```

Figura 2.10: Tabela de encaminhamento do Servidor A

Entradas na Bela:

10.0.7.0/24: rota responsável por encaminhar o tráfego destinado a esta sub rede, a que este dispositivo pertence

0.0.0.0/0: rota default, ou seja rota onde todo o tráfego sem regra de encaminhamento específico segue, que é reencaminhado para o dispositivo 10.0.7.1

```

root@Bela:/tmp/pycore.44587/Bela.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0          10.0.7.1       0.0.0.0         UG      0 0        0 eth0
10.0.7.0         0.0.0.0        255.255.255.0   U       0 0        0 eth0
root@Bela:/tmp/pycore.44587/Bela.conf#

```

Figura 2.11: Tabela de encaminhamento da Bela

## 2.2.2 Alínea b

b. Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema, por exemplo, ps -ax ou equivalente).

Como pode ser observado nas imagens abaixo, o Servidor A tem processos de ospf e zebra a serem executados, processos estes que implementam encaminhamento dinâmico, por outro lado o computador Bela não apresenta nenhum destes processos, logo pode-se concluir que utiliza encaminhamento estático.

```

root@RA:/tmp/pycore.39609/RA.conf# ps -ax
  PID TTY          STAT TIME COMMAND
    1 ?            S      0:00 vncodet -v -c /tmp/pycore.39609/RA -l /tmp/pycore.39
   87 ?            Ss     0:00 /usr/local/sbin/zebra -d
   93 ?            Ss     0:00 /usr/local/sbin/ospf6d -d
   97 ?            Ss     0:00 /usr/local/sbin/ospfd -d
  105 pts/2        Ss     0:00 /bin/bash
  112 pts/2        R+     0:00 ps -ax
root@RA:/tmp/pycore.39609/RA.conf#

```

Figura 2.12: Processos Servidor A

```

root@Bela:/tmp/pycore.39609/Bela.conf# ps -xa
  PID TTY          STAT TIME COMMAND
    1 ?            S      0:00 vncodet -v -c /tmp/pycore.39609/Bela -l /tmp/pycore.
   20 pts/2        Ss     0:00 /bin/bash
   27 pts/2        R+     0:00 ps -xa

```

Figura 2.13: Processos Bela

### 2.2.3 Alínea c

c. Admita que, por questões administrativas, a rota por defeito (0.0.0.0 ou default) deve ser retirada definitivamente da tabela de encaminhamento do servidor SA. Use o comando `route delete` para o efeito. Que implicações tem esta medida para os utilizadores da LEI-RC que acedem ao servidor. Justifique.

Após a remoção da 0.0.0.0 do servidor A, deixa de ser possível a comunicação com este servidor de dispositivos fora da sua rede local. Isto deve-se ao facto que de a única regra presente neste servidor ser 10.0.7.0/24, que define a sua rede local, assim sendo o servidor A não sabe como lidar com datagramas que sejam destinados a IP fora desta rede, o que faz com que este deixe de saber enviar datagramas para fora da sua rede.

### 2.2.4 Alínea d

d. Não volte a repor a rota por defeito. Adicione todas as rotas estáticas necessárias para restaurar a conectividade para o servidor SA, por forma a contornar a restrição imposta na alínea c). Utilize para o efeito o comando `route add` e registe os comandos que usou.

Os comandos usados para restaurar a conectividade do Servidor A foi: `route add -net 10.0.0.0 gw 10.0.7.1 netmask 255.255.0.0`

A rota que foi adicionada representa o aglomerado das sub redes presentes na rede, restaurando a capacidade do Servidor A comunicar com o resto da rede.

### 2.2.5 Alínea e

e. Teste a nova política de encaminhamento garantindo que o servidor está novamente acessível, utilizando para o efeito o comando `ping`. Registe a nova tabela de encaminhamento do servidor.

Na imagem seguinte encontra-se um exemplo de ping do Servidor A para o Servidor C de forma a demonstrar que este se encontra novamente acessível.

```

root@rt:/tmp/pycore.44587/R.conf# ping 10.0.9.20 -c 5
PING 10.0.9.20 (10.0.9.20) 56(84) bytes of data:
64 bytes from 10.0.9.20: icmp_seq=1 ttl=62 time=5.88 ms
64 bytes from 10.0.9.20: icmp_seq=2 ttl=62 time=9.37 ms
64 bytes from 10.0.9.20: icmp_seq=3 ttl=62 time=2.53 ms
64 bytes from 10.0.9.20: icmp_seq=4 ttl=62 time=0.864 ms
64 bytes from 10.0.9.20: icmp_seq=5 ttl=62 time=11.2 ms

--- 10.0.9.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4063ms
rtt min/avg/max/mdev = 0.864/5.375/11.237/3.929 ms
root@rt:/tmp/pycore.44587/R.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask        Flags   MSS Window  irtt Iface
10.0.0.0          10.0.7.1       255.255.0.0    UG      0 0         0 eth0
10.0.7.0          0.0.0.0        255.255.255.0  U       0 0         0 eth0

```

Figura 2.14: Tabela de encaminhamento do Servidor A e ping para Servidor C

## 2.3 Exercício 3

### 2.3.1 Alínea 1

Considere que dispõe apenas do endereço de rede IP 192.168.XXX.128/25, em que XXX é o decimal correspondendo ao seu número de grupo (PLXX). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo as redes de acesso externo e backbone inalteradas), sabendo que o número de departamentos pode vir a aumentar no curto prazo. Atribua endereços às interfaces dos vários sistemas envolvidos. Assuma que todos os endereços de sub-redes são usáveis. Justifique as opções tomadas no planejamento.

Como o nosso número de grupo é o 131, o IP disponível para endereçamento passa a ser 192.168.131.128/25. Como a máscara usada é /25 sabemos que os primeiros 25 bits de rede são fixos e por isso a aplicação do subnetting tem de recorrer apenas aos restantes 7 bits. Estes 7 bits necessitam ainda de ser divididos, uma vez que parte será usada para identificar as subredes e o restante para identificar os diferentes hosts.

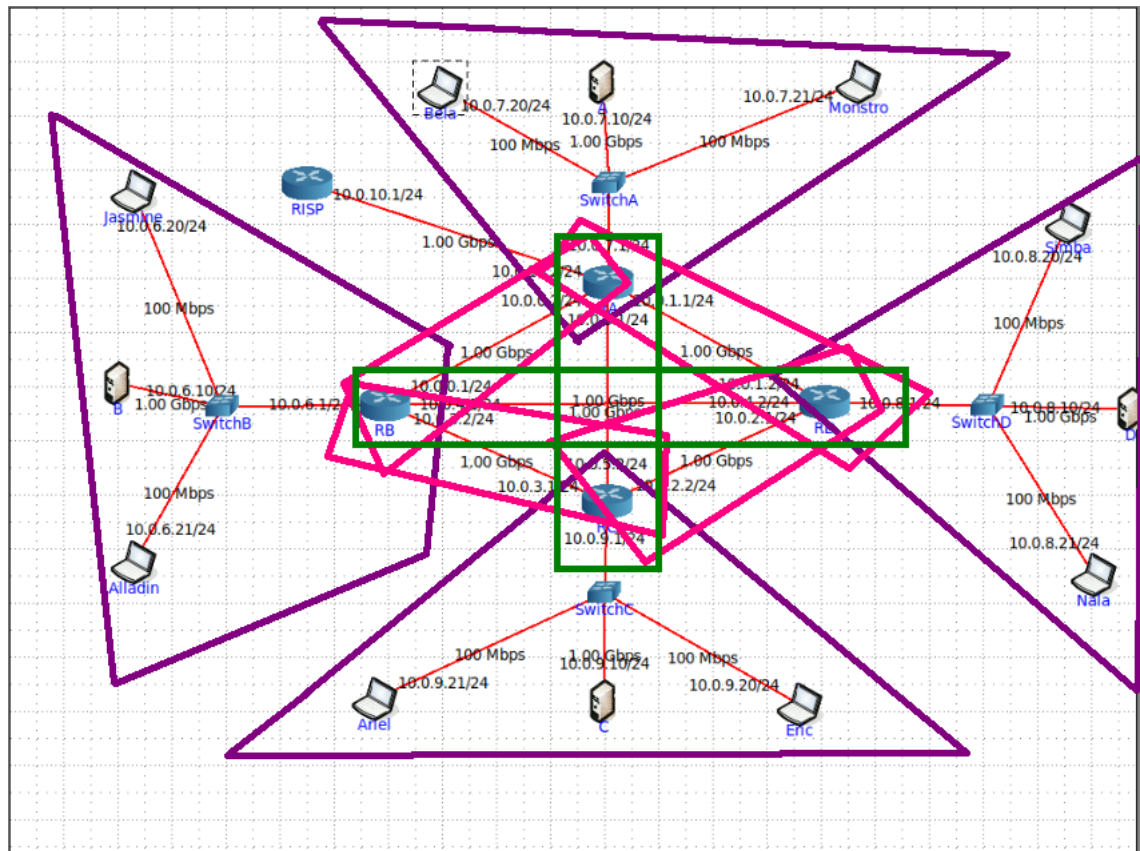


Figura 2.15: Identificação das diferentes subredes existentes

Como é possível verificar na Figura 2.15 na topologia em estudo existem 10 subredes (Correspondentes ao departamentos[4] e às ligações entre os diferentes routers[6]). Sabendo o número total de subredes existente, passa então a ser possível inferir o número de bits necessários para as definir no caso 4, já que  $2^4 = 16$ . Após este cálculo, é então possível atribuir a gama de endereços disponível para cada subrede, procedemos então à atribuição de endereços.

*SubredeA*  $\rightarrow$  0000  $\rightarrow$  192.168.131.128

*SubredeB*  $\rightarrow$  0001  $\rightarrow$  192.168.131.136

*SubredeC*  $\rightarrow$  0010  $\rightarrow$  192.168.131.144

*SubredeD*  $\rightarrow$  0011  $\rightarrow$  192.168.131.152

*SubredeRA – RB*  $\rightarrow$  0100  $\rightarrow$  192.168.131.160



*SubredeRA – RC* → 0101 → 192.168.131.168

*SubredeRA – RD* → 0110 → 192.168.131.176

*SubredeRB – RC* → 0111 → 192.168.131.184

*SubredeRB – RD* → 1000 → 192.168.131.192

*SubredeRC – RD* → 1001 → 192.168.131.200

**Nota:** R\* representa o router do departamento \*

Por fim, resta apenas escolher endereços para hosts e routers pertencentes a estes intervalos de IP's, tendo em conta a subrede a que pertencem e atribuindo sempre o primeiro endereço da gama ao router e os consecutivos aos hosts. Deste processo de atribuição resulta a topologia presente na Figura 2.16.

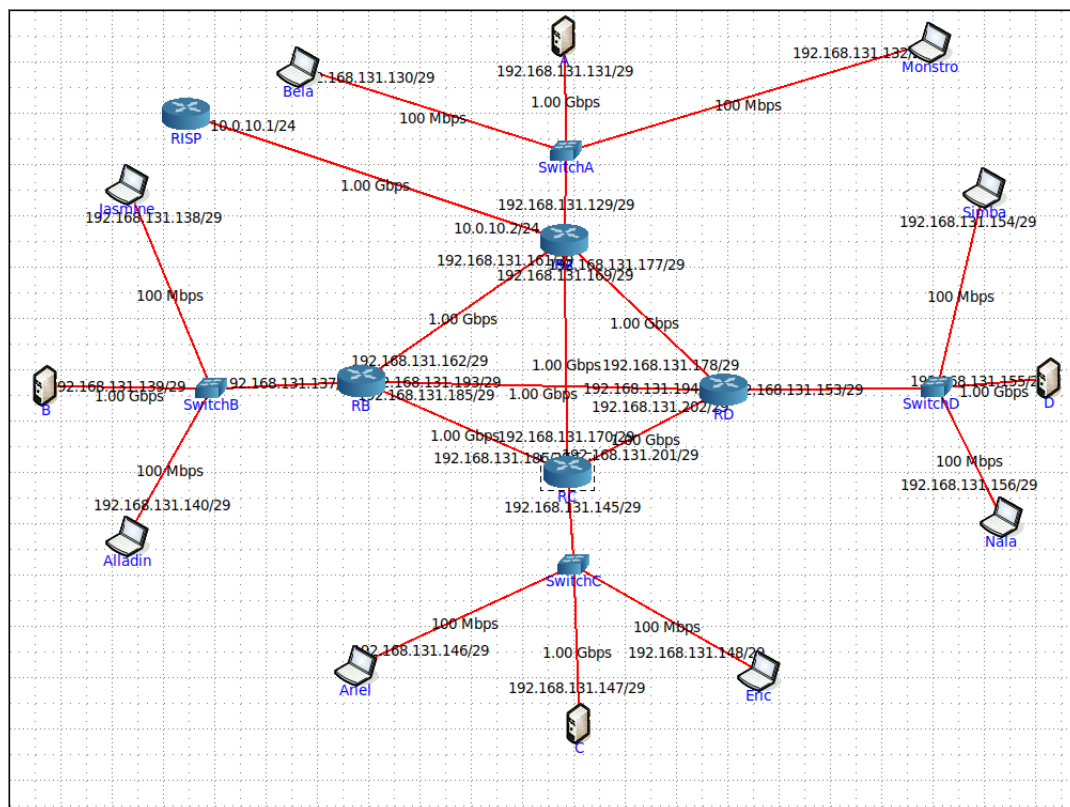


Figura 2.16: Subnetting realizado no core

### 2.3.2 Alínea 2

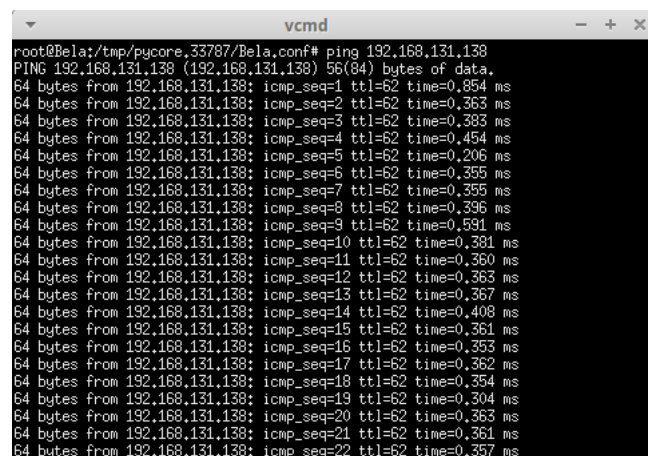
Qual a máscara de rede que usou (em formato decimal)? Quantos hosts IP pode interligar em cada departamento? Quantos prefixos de sub-rede ficam disponíveis para uso futuro? Justifique.

Foi utilizada uma máscara de rede /29 (255.255.255.248), uma vez que de acordo com a máscara inicial, 25 bits são usados para identificar a rede, e dos restantes 7 bits são utilizados 4 para identificar as subredes. Deste forma, conseguimos identificar 6 hosts ( $2^3 - 2$  já que possuímos 3 bits disponíveis e existem 2 endereços reservados), 16 subredes ( $2^4$ ) e já que na topologia existem 10 subredes a serem definidas sobram apenas 6 prefixos para atribuir no futuro.

### 2.3.3 Alínea 3

Verifique e garanta que a conectividade IP interna na rede local LEI-RC é mantida. No caso de não existência de conectividade, reveja a atribuição de endereços efetuada e eventuais erros de encaminhamento por forma a realizar as correções necessárias. Explique como procedeu.

Para realizar o teste de conexão serão enviados vários pacotes entre as diferentes subredes através do comando ping. Os resultados obtidos encontram-se nas figuras seguintes:



```
vcmd
root@Bela:/tmp/pycone_33787/Bela.conf# ping 192.168.131.138
PING 192.168.131.138 (192.168.131.138) 56(84) bytes of data:
64 bytes from 192.168.131.138: icmp_seq=1 ttl=62 time=0.854 ms
64 bytes from 192.168.131.138: icmp_seq=2 ttl=62 time=0.363 ms
64 bytes from 192.168.131.138: icmp_seq=3 ttl=62 time=0.383 ms
64 bytes from 192.168.131.138: icmp_seq=4 ttl=62 time=0.454 ms
64 bytes from 192.168.131.138: icmp_seq=5 ttl=62 time=0.206 ms
64 bytes from 192.168.131.138: icmp_seq=6 ttl=62 time=0.355 ms
64 bytes from 192.168.131.138: icmp_seq=7 ttl=62 time=0.355 ms
64 bytes from 192.168.131.138: icmp_seq=8 ttl=62 time=0.396 ms
64 bytes from 192.168.131.138: icmp_seq=9 ttl=62 time=0.591 ms
64 bytes from 192.168.131.138: icmp_seq=10 ttl=62 time=0.381 ms
64 bytes from 192.168.131.138: icmp_seq=11 ttl=62 time=0.360 ms
64 bytes from 192.168.131.138: icmp_seq=12 ttl=62 time=0.363 ms
64 bytes from 192.168.131.138: icmp_seq=13 ttl=62 time=0.367 ms
64 bytes from 192.168.131.138: icmp_seq=14 ttl=62 time=0.408 ms
64 bytes from 192.168.131.138: icmp_seq=15 ttl=62 time=0.361 ms
64 bytes from 192.168.131.138: icmp_seq=16 ttl=62 time=0.353 ms
64 bytes from 192.168.131.138: icmp_seq=17 ttl=62 time=0.362 ms
64 bytes from 192.168.131.138: icmp_seq=18 ttl=62 time=0.354 ms
64 bytes from 192.168.131.138: icmp_seq=19 ttl=62 time=0.304 ms
64 bytes from 192.168.131.138: icmp_seq=20 ttl=62 time=0.363 ms
64 bytes from 192.168.131.138: icmp_seq=21 ttl=62 time=0.361 ms
64 bytes from 192.168.131.138: icmp_seq=22 ttl=62 time=0.357 ms
```

Figura 2.17: Ping de Bela para Jasmine (SR A - SR B)

```
vcmd
root@Bela:/tmp/pycore.33787/Bela.conf# ping 192.168.131.146 -c 5
PING 192.168.131.146 (192.168.131.146) 56(84) bytes of data:
64 bytes from 192.168.131.146: icmp_seq=1 ttl=62 time=0.605 ms
64 bytes from 192.168.131.146: icmp_seq=2 ttl=62 time=0.763 ms
64 bytes from 192.168.131.146: icmp_seq=3 ttl=62 time=0.508 ms
64 bytes from 192.168.131.146: icmp_seq=4 ttl=62 time=0.344 ms
64 bytes from 192.168.131.146: icmp_seq=5 ttl=62 time=0.354 ms

--- 192.168.131.146 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4090ms
rtt min/avg/max/mdev = 0.344/0.514/0.763/0.157 ms
root@Bela:/tmp/pycore.33787/Bela.conf#
```

Figura 2.18: Ping de Bela para Ariel (SR A - SR C)

```
vcmd
root@Bela:/tmp/pycore.33787/Bela.conf# ping 192.168.131.154 -c 5
PING 192.168.131.154 (192.168.131.154) 56(84) bytes of data:
64 bytes from 192.168.131.154: icmp_seq=1 ttl=62 time=1.74 ms
64 bytes from 192.168.131.154: icmp_seq=2 ttl=62 time=0.343 ms
64 bytes from 192.168.131.154: icmp_seq=3 ttl=62 time=0.484 ms
64 bytes from 192.168.131.154: icmp_seq=4 ttl=62 time=0.431 ms
64 bytes from 192.168.131.154: icmp_seq=5 ttl=62 time=0.123 ms

--- 192.168.131.154 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4063ms
rtt min/avg/max/mdev = 0.123/0.624/1.743/0.572 ms
root@Bela:/tmp/pycore.33787/Bela.conf#
```

Figura 2.19: Ping de Bela para Simba (SR A - SR D)

```
vcmd
root@Jasmine:/tmp/pycore.33787/Jasmine.conf# ping 192.168.131.146 -c 5
PING 192.168.131.146 (192.168.131.146) 56(84) bytes of data:
64 bytes from 192.168.131.146: icmp_seq=1 ttl=62 time=1.81 ms
64 bytes from 192.168.131.146: icmp_seq=2 ttl=62 time=0.369 ms
64 bytes from 192.168.131.146: icmp_seq=3 ttl=62 time=0.370 ms
64 bytes from 192.168.131.146: icmp_seq=4 ttl=62 time=0.203 ms
64 bytes from 192.168.131.146: icmp_seq=5 ttl=62 time=0.444 ms

--- 192.168.131.146 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4071ms
rtt min/avg/max/mdev = 0.203/0.640/1.814/0.592 ms
root@Jasmine:/tmp/pycore.33787/Jasmine.conf#
```

Figura 2.20: Ping de Jasmine para Ariel (SR B - SR C)

```
vcmd
root@Jasmine:/tmp/pycore.33787/Jasmine.conf# ping 192.168.131.154 -c 5
PING 192.168.131.154 (192.168.131.154) 56(84) bytes of data:
64 bytes from 192.168.131.154: icmp_seq=1 ttl=62 time=1.22 ms
64 bytes from 192.168.131.154: icmp_seq=2 ttl=62 time=0.333 ms
64 bytes from 192.168.131.154: icmp_seq=3 ttl=62 time=0.386 ms
64 bytes from 192.168.131.154: icmp_seq=4 ttl=62 time=0.384 ms
64 bytes from 192.168.131.154: icmp_seq=5 ttl=62 time=0.351 ms

--- 192.168.131.154 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4081ms
rtt min/avg/max/mdev = 0.333/0.534/1.218/0.342 ms
root@Jasmine:/tmp/pycore.33787/Jasmine.conf#
```

Figura 2.21: Ping de Jasmine para Simba (SR B - SR D)

```
vcmd
root@Ariel:/tmp/pycore.33787/Ariel.conf# ping 192.168.131.154 -c 5
PING 192.168.131.154 (192.168.131.154) 56(84) bytes of data:
64 bytes from 192.168.131.154: icmp_seq=1 ttl=62 time=1.24 ms
64 bytes from 192.168.131.154: icmp_seq=2 ttl=62 time=0.403 ms
64 bytes from 192.168.131.154: icmp_seq=3 ttl=62 time=0.350 ms
64 bytes from 192.168.131.154: icmp_seq=4 ttl=62 time=0.371 ms
64 bytes from 192.168.131.154: icmp_seq=5 ttl=62 time=0.373 ms

--- 192.168.131.154 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4078ms
rtt min/avg/max/mdev = 0.350/0.548/1.243/0.347 ms
root@Ariel:/tmp/pycore.33787/Ariel.conf#
```

Figura 2.22: Ping de Ariel para Simba (SR C - SR D)

Como o comando Ping foi bem sucedido em todos os testes, é então possível assumir que existe conectividade em toda a topologia.

## Capítulo 3

# Conclusão

O presente trabalho prático permitiu a exploração e consolidação dos conhecimentos relativos às temáticas de Datagramas IP e Fragmentação e Endereçamento e Encaminhamento IP, assim como utilização de software como o Wireshark e Core. Também foram adquiridos conhecimentos relativos à utilização de comandos trace-route e ping.

Deste modo, através da primeira parte do trabalho prático, foram cimentados os conhecimentos relativos ao funcionamento e casos de uso da fragmentação de datagramas, sendo possível analisar as diferentes soluções que são utilizadas dependendo da ocasião, das flags e das opções escolhidas.

Na segunda parte, através de uma topologia mais complexa, foi possível a análise e manipulação de tabelas de encaminhamento, máscaras de rede e subredes.

Em suma, pensa-se que o trabalho desenvolvido atinge os objetivos propostos, uma vez que a realização do mesmo facilitou a compreensão e expansão dos conhecimentos relativos às temáticas abordadas nas aulas Teóricas da Unidade Curricular.