Universidade do Minho

Licenciatura em Engenharia Informática

Redes de Computadores

**Grupo 135**

# TP2: Protocolo IPv4

Joana Alves (A93290)

João Machado (A89510)

Rui Armada (A90468)

Abril 2022

# Parte I

# 1 Questões e Respostas

## 1.1 Questão 1 - Topologia Core

De acordo com as instruções presentes no enunciado, apresentamos de seguida a topologia construída, tendo em conta a alteração do tempo de propagação para 10 ms:

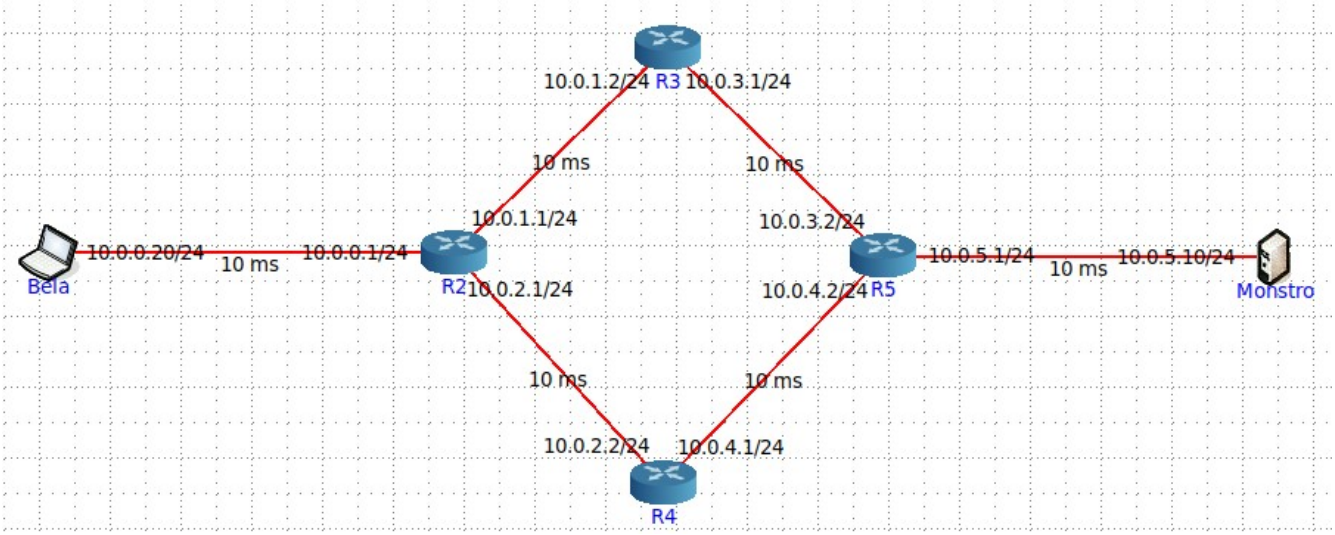



Figura 1: Topologia (Questão 1).

**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**

```
root@Bela:/tmp/pycore.34999/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)  41.232 ms  41.195 ms  41.191 ms
 2  10.0.1.2 (10.0.1.2)  61.868 ms  61.866 ms  61.863 ms
 3  10.0.3.2 (10.0.3.2)  81.992 ms  81.990 ms  81.987 ms
 4  10.0.5.10 (10.0.5.10)  124.138 ms  124.135 ms  124.132 ms
root@Bela:/tmp/pycore.34999/Bela.conf#
```

Figura 2: Print da *shell* no *host* Bela.

**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.**

| Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| 1.185969729 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=6/1536, ttl=2 (no response found!) |
| 1.185970113 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=7/1792, ttl=3 (no response found!) |
| 1.185970497 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=8/2048, ttl=3 (no response found!) |
| 1.185970990 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=9/2304, ttl=3 (no response found!) |
| 1.185971375 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=10/2560, ttl=4 (reply in 39) |
| 1.185971763 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=11/2816, ttl=4 (reply in 40) |
| 1.185972461 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=12/3072, ttl=4 (reply in 41) |
| 1.185973505 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=13/3328, ttl=5 (reply in 42) |
| 1.185973894 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=14/3584, ttl=5 (reply in 43) |
| 1.185974277 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=15/3840, ttl=5 (reply in 44) |
| 1.185974662 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=16/4096, ttl=6 (reply in 45) |
| 1.206170814 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.206174897 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.206175508 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.206765251 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=17/4352, ttl=6 (reply in 46) |
| 1.206771362 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=18/4608, ttl=6 (reply in 47) |
| 1.206774684 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=19/4864, ttl=7 (reply in 48) |
| 1.227506185 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.227510248 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.227511028 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.227749392 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=20/5120, ttl=7 (reply in 49) |
| 1.227755966 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=21/5376, ttl=7 (reply in 50) |
| 1.227759358 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=22/5632, ttl=8 (reply in 51) |
| 1.247740650 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.247747614 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.247748760 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 1.248013196 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=23/5888, ttl=8 (reply in 52) |
| 1.248019504 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=24/6144, ttl=8 (reply in 53) |
| 1.248023303 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0023, seq=25/6400, ttl=9 (reply in 54) |
| 1.288887873 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x0023, seq=10/2560, ttl=61 (request in 14) |

Figura 3: Tráfego ICMP enviado e recebido pelo *host* Bela.

O comando *traceroute* envia, através do *host*, vários datagramas ICMP variando o valor de TTL (*Time To Live*) para assim obter a rota desde a origem até ao IP destino (especificado no comando *traceroute*). A variável TTL indica o número de saltos que o datagrama pode fazer dentro da rede antes de ser considerado inválido, sendo este valor decrementado sempre que executa um salto. O seu comportamento, de forma resumida, pode ser descrito como o envio sequencial de conjuntos de um ou mais datagramas ICMP com o mesmo TTL, iniciando este valor a 1 e incrementando a cada iteração. Um *host* ao verificar que o valor do TTL de um datagrama é 0 envia como resposta uma mensagem de controlo (datagrama ICMP) com a informação de 'TTL *exceeded*'. Assim, o *host* que enviou os datagramas iniciais vai obtendo, ao longo das iterações, uma noção cada vez mais completa da rota até ao destino.

No caso concreto deste exercício, podemos, através da Figura 3, verificar o envio destes conjuntos de três datagramas ICMP logo a partir da primeira linha, começando, tal como dito, no valor TTL=1 e sendo incrementado a cada iteração.

Apresentamos, de seguida, uma tabela com a correspondência entre os valores iniciais de TTL dos datagramas e os *hosts* que os receberam quando estes atingiram o valor 0, ou seja, excederam o seu número de saltos na rede:

| TTL | Host Final |
|---|---|
| 1 | 10.0.0.1 |
| 2 | 10.0.1.2 |
| 3 | 10.0.3.2 |

Como podemos verificar pela tabela, os datagramas enviados com TTL=1 ficaram inválidos no *host* 10.0.0.1, os datagramas com TTL=2 no *host* 10.0.1.2 e, por fim, os datagramas com TTL=3 foram invalidados pelo *host* 10.0.3.2.

Podemos reparar que os datagramas com TTL superior a 3 não registaram nenhuma resposta de 'TTL *exceeded*' uma vez que a partir desse valor os pacotes conseguem chegar a qualquer nodo da rede, inclusive o nodo objetivo (10.0.5.10).

**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.**

O valor mínimo do campo TTL necessário para alcançar o servidor *Monstro* será **4**, uma vez que valores inferiores a 3 (inclusive) têm resposta de tempo de vida excedido (como visto na alínea anterior). Podemos verificar isto pela não receção de datagramas ICMP por parte do *host* Bela a partir dos valores de TTL superiores ou iguais a 4, ou pelo cálculo direto através da observação da topologia, onde verificamos que são necessários, no mínimo, quatro saltos para atingir o servidor.

**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.41619/Bela.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)  20.466 ms  20.450 ms  20.446 ms  20.444 ms  20.442 ms  20.439 ms * * * *
 2  10.0.1.2 (10.0.1.2)  41.758 ms  41.756 ms  41.754 ms  41.751 ms  41.749 ms  41.745 ms * * * *
 3  10.0.3.2 (10.0.3.2)  61.995 ms  61.993 ms  61.461 ms  61.449 ms  61.447 ms  61.444 ms * * * *
 4  10.0.5.10 (10.0.5.10)  82.263 ms  82.262 ms  81.318 ms  81.306 ms  81.967 ms  81.956 ms  81.952 ms
 81.951 ms  82.746 ms  82.734 ms
root@Bela:/tmp/pycore.41619/Bela.conf#
```

Figura 4: Aplicação do comando 'traceroute -I 10.0.5.10 -q 10'.

De acordo com o resultado do comando pedido, retiramos os valores atingidos no acesso ao servidor *Monstro* a partir da linha numerada com o valor 4. Assim, calculamos a média destes mesmos valores, tendo obtido o seguinte resultado:

$$\frac{82.263 + 82.262 + 81.318 + 81.306 + 81.967 + 81.956 + 81.952 + 81.951 + 82.746 + 82.734}{10} = 81.946ms$$

**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?**

O cálculo torna-se difícil devido às exigências de sincronização, uma vez que *delay* é uma métrica que exige sincronização de relógios. *One-Way Delay*, tal como o nome indica, só funciona num sentido, mas precisa de ter acesso a algumas informações para poder ser calculado, nomeadamente o registo de quando o pacote foi enviado (*timestamp*). Em consequência, é imperativo que os relógios estejam sincronizados, caso contrário esta informação torna-se irrelevante. O *delay* de ida é diferente do de volta pois o percurso percorrido pode não ser idêntico, por conseguinte, o valor médio de atraso num sentido não pode ser calculado com precisão dividindo o RTT por dois.

## 1.2 Questão 2 - *Traceroute*

```
joana joana ~ traceroute -I marco.uminho.pt
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 60 byte packets
 1  _gateway (172.26.254.254)  3.169 ms  3.312 ms  3.505 ms
 2  172.16.2.1 (172.16.2.1)  5.750 ms  5.962 ms  6.169 ms
 3  172.16.115.252 (172.16.115.252)  6.375 ms  6.583 ms  8.251 ms
 4  marco.uminho.pt (193.136.9.240)  7.163 ms  8.234 ms  8.183 ms
```

Figura 5: Resultado do comando *traceroute* na máquina nativa.

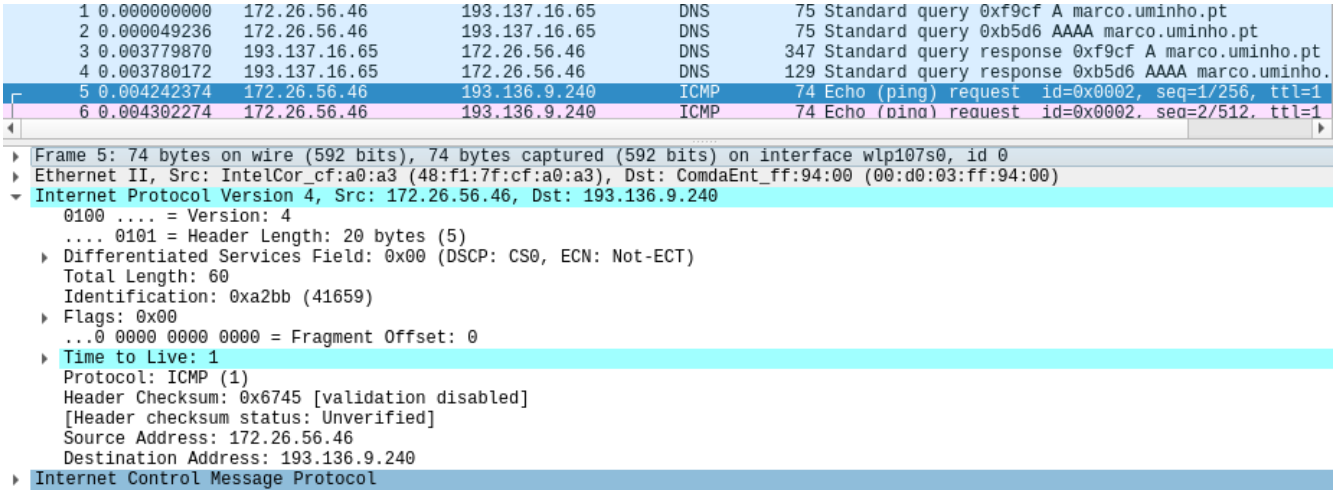



Figura 6: Informação da primeira mensagem ICMP enviada pela máquina.

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

Como se pode observar na Figura 6, na informação da mensagem ICMP enviada pela máquina nativa, no parâmetro *Source Address*, a interface que foi utilizada para realizar este teste possui o endereço IP de **172.26.56.46**.

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

Como podemos verificar pela Figura 6, o valor do campo *Protocol* é 1, correspondendo ao protocolo ICMP. O conteúdo deste campo especifica o protocolo que está a ser encapsulado pelo IP para assim permitir a descodificação do mesmo pelas camadas devidas.

### 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 um total de 20 *bytes*. Uma vez que o *payload* se calcula subtraindo o tamanho do cabeçalho ao tamanho total do datagrama, o campo de dados tem 40 (60-20) *bytes*.

**d. O datagrama IP foi fragmentado? Justifique.**

O datagrama não foi fragmentado uma vez que a *flag* de não fragmentação não está definida no cabeçalho no campo *Flag*. Para além disso, podemos verificar que o valor do *offset* é 0, logo estamos perante o datagrama original, sem fragmentação.

**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.**

| | | | | | |
|---|---|---|---|---|---|
| 5 | 0.004242374 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=1/256, ttl=1 (no response found!) |
| 6 | 0.004302274 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=2/512, ttl=1 (no response found!) |
| 7 | 0.004325857 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=3/768, ttl=1 (no response found!) |
| 8 | 0.004349947 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=4/1024, ttl=2 (no response found!) |
| 9 | 0.004368346 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=5/1280, ttl=2 (no response found!) |
| 10 | 0.004386482 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=6/1536, ttl=2 (no response found!) |
| 11 | 0.004408433 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=7/1792, ttl=3 (no response found!) |
| 12 | 0.004426652 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=8/2048, ttl=3 (no response found!) |
| 13 | 0.004446026 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=9/2304, ttl=3 (no response found!) |
| 14 | 0.004470241 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=10/2560, ttl=4 (reply in 33) |
| 15 | 0.004514346 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=11/2816, ttl=4 (reply in 35) |
| 16 | 0.004552836 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=12/3072, ttl=4 (reply in 36) |
| 17 | 0.004589173 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=13/3328, ttl=5 (reply in 37) |
| 18 | 0.004609745 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=14/3584, ttl=5 (reply in 38) |
| 19 | 0.004628864 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=15/3840, ttl=5 (reply in 39) |
| 20 | 0.004651809 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=16/4096, ttl=6 (reply in 40) |
| 24 | 0.008143685 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=17/4352, ttl=6 (reply in 41) |
| 25 | 0.008246717 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=18/4608, ttl=6 (reply in 42) |
| 26 | 0.008280842 | 172.26.56.46 | 193.136.9.240 | ICMP | 74 Echo (ping) request id=0x0002, seq=19/4864, ttl=7 (reply in 43) |

Figura 7: Sequência de mensagens ICMP enviadas pela máquina.

```
▾ Internet Protocol Version 4, Src: 172.26.56.46, 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: 0xa2bb (41659)
  ▸ Flags: 0x00
    ...0 0000 0000 0000 = Fragment Offset: 0
  ▸ Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0x6745 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.56.46
    Destination Address: 193.136.9.240
▸ Internet Control Message Protocol
```

Figura 8: Primeiro pacote com TTL=1.

```
▾ Internet Protocol Version 4, Src: 172.26.56.46, 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: 0xa2be (41662)
  ▸ Flags: 0x00
    ...0 0000 0000 0000 = Fragment Offset: 0
  ▸ Time to Live: 2
    Protocol: ICMP (1)
    Header Checksum: 0x6642 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.56.46
    Destination Address: 193.136.9.240
▸ Internet Control Message Protocol
```

Figura 9: Primeiro pacote com TTL=2.

Analisando os pacotes da sequência de mensagens ICMP enviadas pela máquina nativa, podemos inferir que alguns campos se alteram ao longo da sequência. Assim, conseguimos distinguir os seguintes campos: ***Identification***, ***Checksum*** e **TTL**. O *checksum* é um parâmetro de controlo de integridade do pacote, sendo, por isso, normal que o seu valor não coincida. Relativamente aos outros dois campos, é explicado na alínea seguinte.

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

O campo de identificação é incrementado sequencialmente e o campo TTL vai incrementando em conjuntos de três pacotes, ou seja, três pacotes são enviados com o mesmo valor (TTL = k) seguidos de outros três com o valor inteiro diretamente acima (TTL = k+1).

**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?**

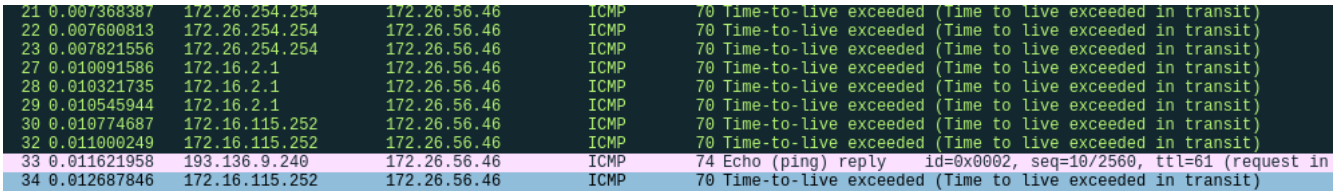



Figura 10: Sequência de mensagens ICMP enviadas pelo endereço destino.

O campo TTL não se mantém constante para todas as mensagens de resposta, tendo um valor diferente para cada *host*. Isto acontece porque como os pacotes são enviados pelos *routers* em que o TTL falha, o número de saltos necessários para atingir a nossa máquina ou definidos como *default* pelos mesmos variam.

## 1.3 Questão 3 - Fragmentação

De acordo com o requisito do enunciado, executamos o comando *traceroute* definindo o tamanho do pacote para 4000 + 135 (nº do grupo) = **4135**.

```
joana  joana  ~  traceroute -I marco.uminho.pt 4135
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 4135 byte packets
 1  _gateway (172.26.254.254)  4.209 ms  6.118 ms  6.458 ms
 2  172.16.2.1 (172.16.2.1)  3.846 ms  4.271 ms  5.788 ms
 3  172.16.115.252 (172.16.115.252)  6.169 ms  7.703 ms  8.649 ms
 4  marco.uminho.pt (193.136.9.240)  10.898 ms  10.872 ms  12.980 ms
```

Figura 11: Resultado do comando *traceroute* com tamanho de pacote definido.

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

```
    7 0.004289274   172.26.56.46        193.136.9.240       ICMP     1189 Echo (ping) request  id=0x0003, seq=1/256, ttl=1 (no response found!)
Frame 7: 1189 bytes on wire (9512 bits), 1189 bytes captured (9512 bits) on interface wlp107s0, id 0
Ethernet II, Src: IntelCor_cf:a0:a3 (48:f1:7f:cf:a0:a3), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.56.46, 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: 1175
    Identification: 0x60c1 (24769)
  ▸ Flags: 0x01
    ...0 1011 1001 0000 = Fragment Offset: 2960
  ▸ Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0xa372 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.56.46
    Destination Address: 193.136.9.240
  ▸ [3 IPv4 Fragments (4115 bytes): #5(1480), #6(1480), #7(1155)]
```

Figura 12: Primeiro pacote ICMP enviado pela máquina.

Foi necessário fragmentar o primeiro pacote visto que o limite da camada de *IP* é de 1500 *bytes* em contraste com o tamanho do pacote a ser enviado (4135 *bytes*), ou seja o MTU é inferior ao tamanho do pacote a ser enviado.

**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?**

```
    5 0.004265392   172.26.56.46        193.136.9.240       IPv4     1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=60c1) [Reassembled in #7]
  ▸ Frame 5: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp107s0, id 0
  ▸ Ethernet II, Src: IntelCor_cf:a0:a3 (48:f1:7f:cf:a0:a3), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  ▾ Internet Protocol Version 4, Src: 172.26.56.46, 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: 0x60c1 (24769)
    ▸ Flags: 0x20, More fragments
      ...0 0000 0000 0000 = Fragment Offset: 0
    ▸ Time to Live: 1
      Protocol: ICMP (1)
      Header Checksum: 0x839f [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 172.26.56.46
      Destination Address: 193.136.9.240
      [Reassembled IPv4 in frame: 7]
  ▸ Data (1480 bytes)
```

Figura 13: Primeiro pacote fragmentado.

O cabeçalho indica-nos que o datagrama foi fragmentado pois tem a *flag* ***More Fragments*** acionada. Para além disto, sabemos que se trata do primeiro fragmento do datagrama original pois o valor no campo ***Fragment Offset*** é 0, ou seja, este datagrama tem um deslocamento de dados igual a 0 relativamente ao datagrama original. Assim, este datagrama tem um tamanho de 1500 *bytes* (20 para cabeçalho e 1480 para dados).

**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?**

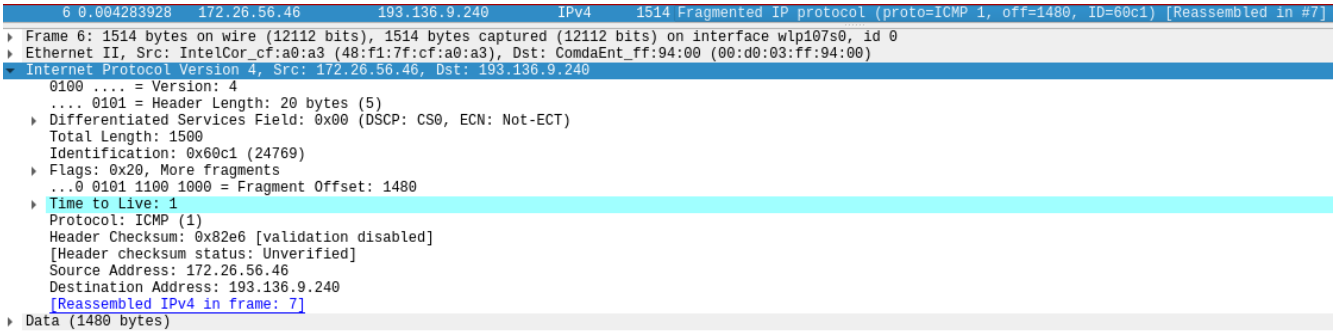



Figura 14: Segundo pacote fragmentado.

Conseguimos verificar, através do valor presente no campo ***Fragment Offset*** do cabeçalho, que não se trata do primeiro fragmento do datagrama, uma vez que esse valor é diferente de 0, sendo, neste caso, 1480, indicando, por isso, um deslocamento de 1480 *bytes* relativamente ao datagrama original. Para além disto, conseguimos denotar que se vão seguir mais fragmentos através do acionamento da *flag* ***More Fragments***.

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

A partir do datagrama original, foram criados três fragmentos como podemos ver pelas Figuras acima. O primeiro fragmento (Figura 13) e o segundo (Figura 14) com um total de 1480 *bytes* de dados. Por fim, o último fragmento (Figura 12) corresponde ao primeiro pacote ICMP enviado pela máquina, com um total de 1155 *bytes* de dados.

**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 se vão alterando ao longo da sequência de fragmentos são *checksum*, *flags*, *offset* e o tamanho do último fragmento relativamente aos anteriores, uma vez que apenas este pode ter ou não o número máximo de dados por pacote. O valor de *checksum*, como referido em alíneas anteriores, serve para verificar a integridade do pacote, no entanto, não tem relação direta com a reconstrução do datagrama original. Já os campos das *flags* e o *offset* permitem fazer a reconstrução do datagrama, na medida em que o *offset* indica a posição do fragmento relativamente aos dados originais, permitindo a ordenação correta dos fragmentos, e as *flags* permitem afirmar se estamos localizados no último pacote ou se ainda existem mais.

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

**Cálculos:**

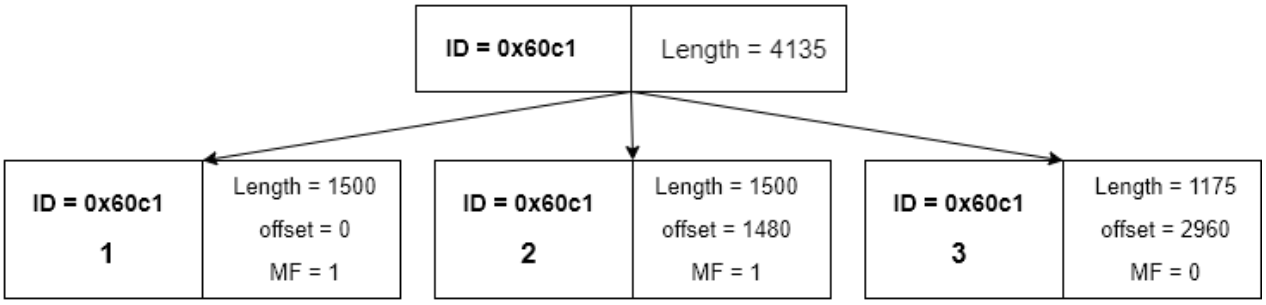
- Tamanho Total = 4135 bytes e MTU = 1500 bytes **=>** é necessário fragmentação
- 4135 - 20 (cabeçalho) = 4115 / 1500 = 2,7... = **3** fragmentos

**Divisão dos Dados:**

Em cada pacote o limite máximo de dados será 1480 uma vez que 20 bytes são necessários para o cabeçalho

| Fragmento | Bytes Enviados |
|---|---|
| 1 | 1480 |
| 2 | 1480 |
| 3 | 1155 |
| **Total** | 4115 |

**Resultado:**




Como podemos verificar pelos cálculos, o datagrama original teria de ser dividido em três fragmentos. Os fragmentos mantêm o mesmo campo de identificação do datagrama original, alterando apenas as *flags* de *More Fragments* (MF) e o valor do *offset* relativamente ao original. A *flag* MF está apresentada com os valores 0 (equivalente ao valor '*not set*') e 1, tendo estes o significado de não acionada e acionada, respetivamente.

De forma adicional, podemos reparar que o *overhead* de cabeçalho aumentou, uma vez que originalmente tínhamos apenas 20 *bytes* de cabeçalho e devido à fragmentação obtivemos um conjunto de três cabeçalhos, ou seja, 60 *bytes*. Como consequência, houve um aumento do número total de *bytes* transferidos na rede, pois no datagrama original seriam 4135 *bytes* em contraste com a fragmentação, onde se registaram no total: 4115 (dados) + 3 * 20 (cabeçalho) = 4175 *bytes*.

Em suma, todos os valores calculados coincidiram com os valores presentes nos campos dos datagramas fragmentados capturados pelo programa.

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

De acordo com os parâmetros do cabeçalho dos pacotes analisados nas alíneas acima, conseguimos deduzir os campos que afetam diretamente a deteção do último fragmento da *stream*, sendo estes a *flag more fragments* e o campo *offset*. O valor do campo *offset* tem de ser diferente de zero para assim concluirmos que, antes de mais, se trata de um datagrama fragmentado. A *flag more fragments* transmite a informação se o fragmento se trata do último ou não. Por último, para o pacote fragmentado corresponder a um fragmento do original que procuramos, o campo de identificação tem de ser igual. Assim, apresentamos a expressão lógica capaz de detetar o último fragmento tendo em conta estes campos do cabeçalho:

$$\text{ID} == \text{original} \quad \&\& \quad \text{More Fragments} == \textit{not set} \quad \&\& \quad \text{offset} > 0$$

De seguida, de forma a testar a nossa expressão, executamos a mesma na secção de *filter* da ferramenta *Wireshark* com a única diferença de não incluirmos a verificação do campo de identificação, obtendo assim todos os últimos fragmentos de todos os datagramas que sofreram fragmentação. Como podemos conferir pela Figura 15, obtivemos o resultado esperado:

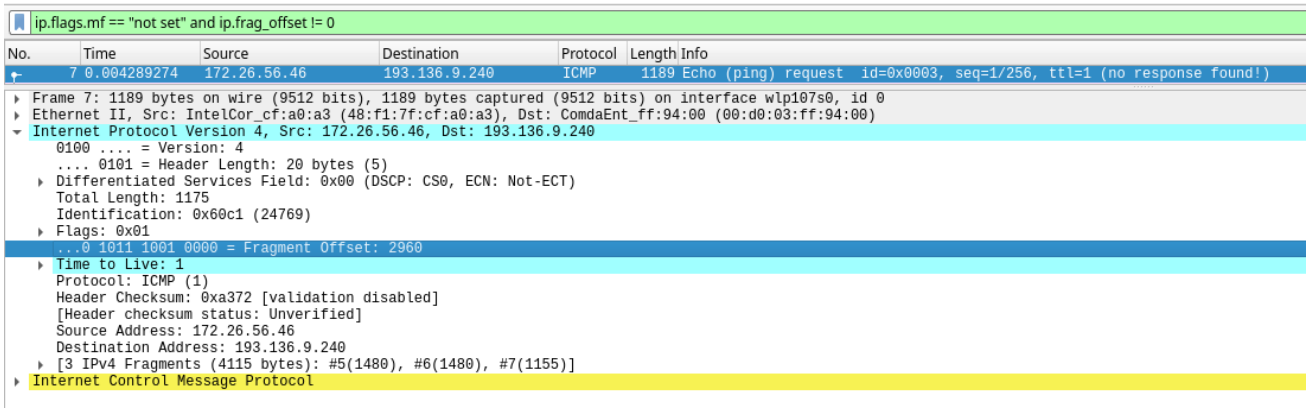



Figura 15: Verificação da expressão na ferramenta *Wireshark*.

# Parte II

# 2 Questões e Respostas

## 2.1 Questão 1 - Topologia LEI-RC

**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.**

Pela Figura 16 conseguimos ter uma visão geral da topologia assim como a divisão entre os vários departamentos. Assim, a todos os endereços é aplicada uma máscara de rede de 25 *bits*, sendo os endereços das sub-redes dos departamentos os seguintes:

- **Departamento A:** 10.0.4.0
- **Departamento B:** 10.0.5.0
- **Departamento C:** 10.0.6.0
- **Departamento D:** 10.0.7.0

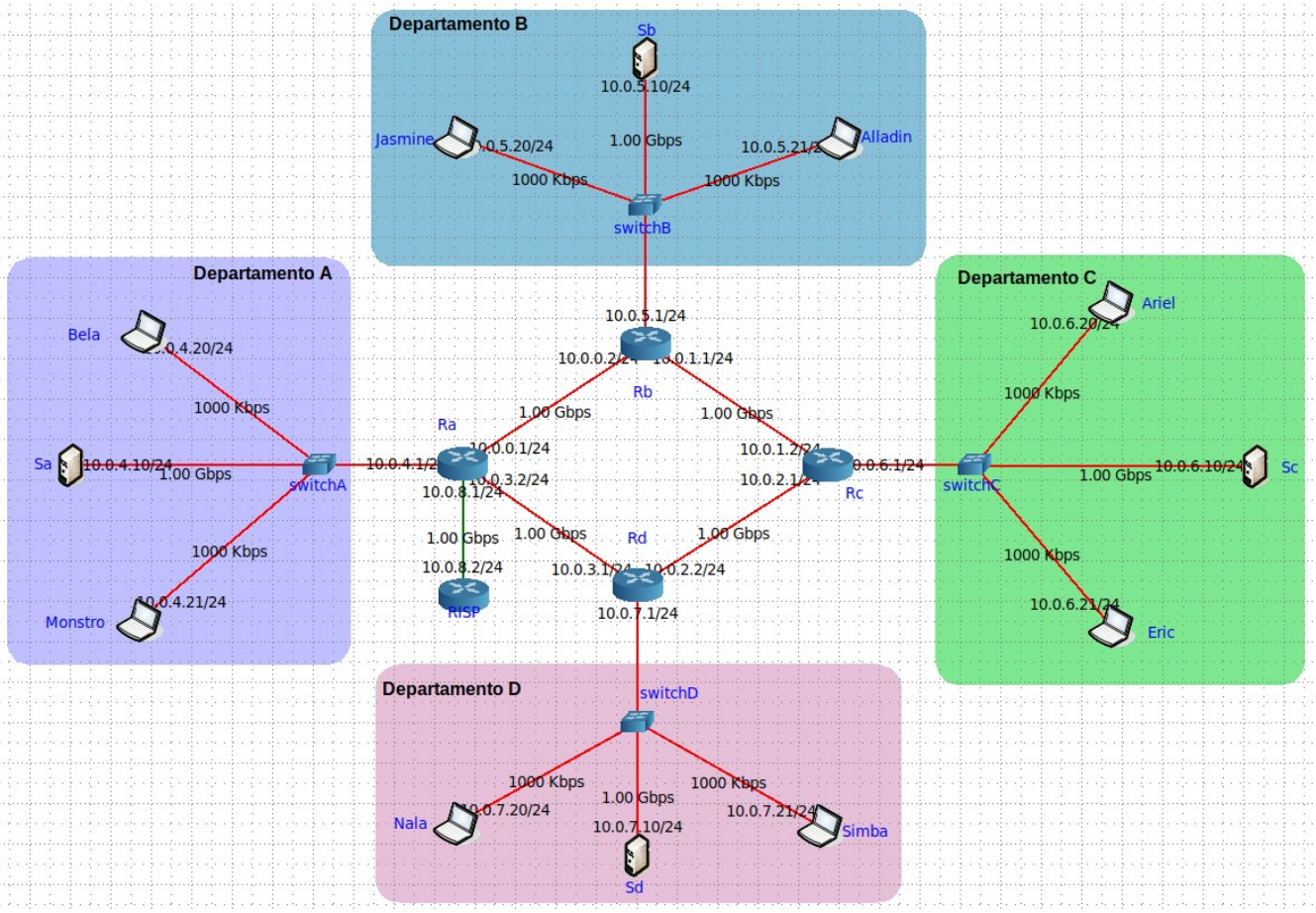



Figura 16: Topologia LEI-RC.

**b. Tratam-se de endereços privados? Porquê?**

Geralmente os endereços privados encontram-se reservados entre intervalos específicos geridos pela *Internet Assigned Numbers Authority (IANA)* e, por essa razão, não podem ser atribuídos de forma arbitrária. Apresentamos, então, alguns intervalos reservados para endereços privados:

- 10.0.0.0 - 10.255.255.255 / 8
- 172.16.0.0 - 172.31.255.255 / 12
- 192.168.0.0 - 192.168.255.255 / 16

Visto que todos os endereços presentes na topologia se encontram incluídos no intervalo de endereços do primeiro ponto, podemos afirmar que os endereços se tratam de endereços privados.

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

Os *switches* tratam-se de dispositivos que simplesmente conectam todos os elementos da rede. Estes atuam como ponte ou unidade de controlo para que os dispositivos possam comunicar entre si. Estes pertencem à camada dois do modelo de referência OSI, ou seja, camada de *Link*. Esta camada está localizada diretamente abaixo da camada de rede (camada três) que funciona sobre endereços IP, no entanto, a camada de *Link* utiliza endereços MAC (endereços físicos) daí não ter sido atribuído endereço IP aos *switches*.

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

Como podemos ver pelas figuras apresentadas abaixo, existe conectividade interna nos vários departamentos uma vez que existe resposta (*echo reply*) ao envio de pacotes (*echo request*) entre dispositivos no mesmo departamento.

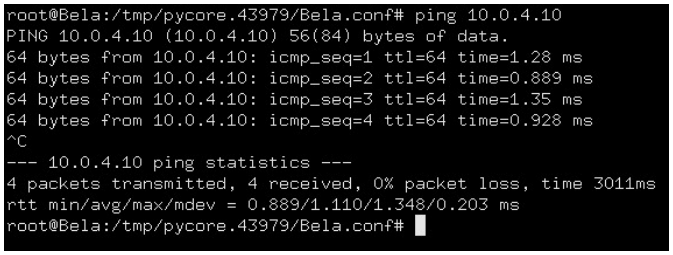



Figura 17: Departamento A (Bela - Sa).

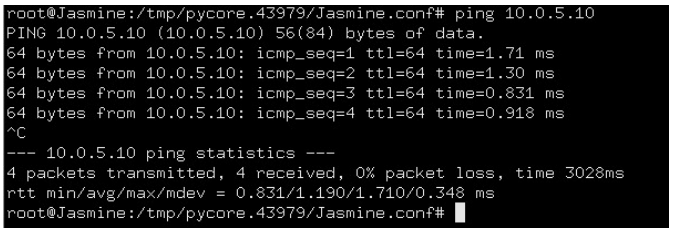



Figura 18: Departamento B (Jasmine -Sb).

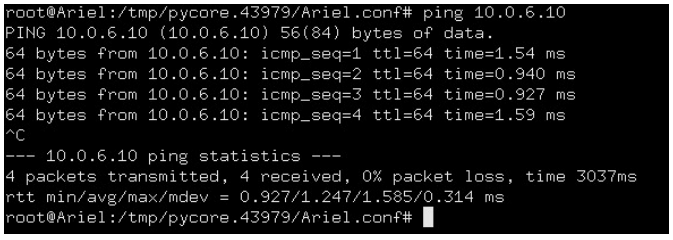



Figura 19: Departamento C (Ariel - Sc).

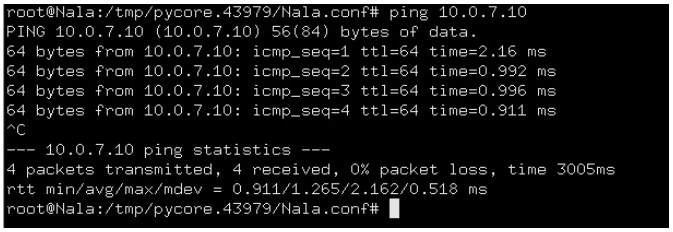



Figura 20: Departamento D (Nala - Sd).

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

Para verificar a conectividade IP entre os vários departamentos, apenas executamos o comando *ping* duas vezes, sendo estas:

- Departamento A → Departamento C (Figura 21)
- Departamento D → Departamento B (Figura 22)

Uma vez que o *ping* do departamento A para o C garante que o tráfego na direção horizontal da topologia está a ser encaminhado corretamente e o *ping* do departamento D ao B garante o bom encaminhamento vertical do tráfego da topologia, conseguimos, assim, garantir a conectividade entre todos os departamentos pois temos garantias do correto funcionamento dos quatro *routers*.

```
root@Monstro:/tmp/pycore.43979/Monstro.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=61 time=1.67 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=61 time=1.29 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=61 time=1.28 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=61 time=0.996 ms
^C
--- 10.0.6.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 0.996/1.307/1.670/0.239 ms
root@Monstro:/tmp/pycore.43979/Monstro.conf#
```

Figura 21: Ping do Departamento A para o C.

```
root@Simba:/tmp/pycore.43979/Simba.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=61 time=1.79 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=1.11 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=1.28 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=61 time=1.77 ms
^C
--- 10.0.5.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.108/1.487/1.789/0.299 ms
root@Simba:/tmp/pycore.43979/Simba.conf#
```

Figura 22: Ping do Departamento D para o B.

**f. Verifique se existe conectividade IP do portátil Bela para o router de acesso Risp.**

Como podemos verificar pela Figura 23, ao executar o comando *ping* do portátil Bela para o *router* de acesso Risp, este obtém resposta provando assim a conectividade entre os mesmos.

```
root@Bela:/tmp/pycore.43979/Bela.conf# ping 10.0.8.2
PING 10.0.8.2 (10.0.8.2) 56(84) bytes of data.
64 bytes from 10.0.8.2: icmp_seq=1 ttl=63 time=2.13 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=63 time=1.41 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=63 time=1.02 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=63 time=1.03 ms
^C
--- 10.0.8.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.017/1.398/2.129/0.450 ms
root@Bela:/tmp/pycore.43979/Bela.conf#
```

Figura 23: Ping Bela - Risp

## 2.2 Questão 2 - Tabelas de Encaminhamento (Bela - Ra)

**a. Execute o comando *netstat -rn* por forma a poder consultar a tabela de encaminhamento unicast (IPV4). Interprete as várias entradas de cada tabela**

```
root@Bela:/tmp/pycore.43979/Bela.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.4.1        0.0.0.0         UG        0 0          0 eth0
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
root@Bela:/tmp/pycore.43979/Bela.conf#
```

Figura 24: Tabela de Encaminhamento do *host* Bela.

```
root@Ra:/tmp/pycore.43979/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        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.2.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth1
10.0.3.0        0.0.0.0         255.255.255.0   U         0 0          0 eth1
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth2
10.0.5.0        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.6.0        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.7.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth1
10.0.8.0        0.0.0.0         255.255.255.0   U         0 0          0 eth3
root@Ra:/tmp/pycore.43979/Ra.conf#
```

Figura 25: Tabela de Encaminhamento do *router* Ra.

Uma tabela de encaminhamento fornece instruções para determinar o próximo salto de um pacote de dados numa rede IP. Como tal, tem de possuir parâmetros que permitam executar esse mesmo encaminhamento. Assim, apresentamos, também em formato tabela, a descrição das várias colunas da tabela de encaminhamento dos dois dispositivos:

| Coluna | Descrição |
|---|---|
| Destination | endereço IP de rede destino |
| Gateway | endereço IP do próximo salto |
| Genmask | máscara a aplicar à coluna Destination |
| Flags* | informação sobre a rota |
| MSS | tamanho máximo de segmentos TCP |
| Window | tamanho *default* da janela TCP |
| irtt | RTT estimado (inicial) |
| Iface | interface de saída |

* Existem várias *flags* disponíveis, no entanto, nas figuras acima, apenas percepcionamos a **U** e **G**. A presença da *flag* U determina que a rota está disponível e a *flag* G indica a utilização da *Gateway*.

Relativamente às entradas das duas tabelas de encaminhamento, estas possuem a informação necessária para os pacotes de dados enviados por estes conseguirem alcançar o seu destino.

Em concreto, o portátil Bela apenas possui duas entradas que por sua vez descrevem a única interface de saída do mesmo, estando a primeira definida como rota por defeito e a segunda como próximo salto caso o destino seja a sua própria rede. Assim, o portátil Bela consegue assegurar o tráfego dos seus pacotes uma vez que, por defeito, redireciona o tráfego para o *router* Ra.

O *router* Ra possui entradas para todas as sub-redes presentes na topologia, permitindo assim encaminhar tráfego para as mesmas. Para além disto, conseguimos denotar que todas as entradas com endereço IP 0.0.0.0 na coluna *Gateway* têm como destino uma sub-rede da qual o *router* já faz parte, isto acontece porque uma vez que o *router* já está integrado na sub-rede, então já não existe *gateway*.

### b. Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico

```
root@Bela:/tmp/pycore.43979/Bela.conf# ps -ax
    PID TTY      STAT   TIME COMMAND
      1 ?        S      0:00 vnoded -v -c /tmp/pycore.43979/Bela -l /tmp/pycore.43979/Bela.log -p
     52 pts/4    Ss     0:00 /bin/bash
     63 pts/4    R+     0:00 ps -ax
root@Bela:/tmp/pycore.43979/Bela.conf#
```

Figura 26: Resultado do comando no *host* Bela.

```
root@Ra:/tmp/pycore.43979/Ra.conf# ps -ax
    PID TTY      STAT   TIME COMMAND
      1 ?        S      0:00 vnoded -v -c /tmp/pycore.43979/Ra -l
     75 ?        Ss     0:00 /usr/local/sbin/zebra -d
     82 ?        Ss     0:00 /usr/local/sbin/ospf6d -d
     86 ?        Ss     0:00 /usr/local/sbin/ospfd -d
     93 pts/2    Ss     0:00 /bin/bash
    101 pts/2    R+     0:00 ps -ax
root@Ra:/tmp/pycore.43979/Ra.conf#
```

Figura 27: Resultado do comando no *router* RA.

Pelas Figuras 26 e 27 conseguimos perceber algumas diferenças nos processos que estão a decorrer em cada um dos dispositivos. A presença de um processo que termina com ***ospf*** significa que o protocoloo OSPF está a ser utilizado. O protocolo OSPF (*Open Shortest Path First*), é um protocolo de descoberta de vizinhos, isto é, os *routers* ou *hosts* depois de garantirem que as suas interfaces estão funcionais, enviam pacotes '*Hello*' para descobrir os dispositivos adjacentes (vizinhos) e as rotas conhecidas pelos mesmos. A utilização do protocolo OSPF garante um encaminhamento dinâmico, uma vez que este, através da comunicação entre os vários dispositivos, define o melhor caminho. Assim, podemos concluir que o portátil Bela não utiliza encaminhamento dinâmico, contrariamente ao *router* Ra que utiliza.

**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.**

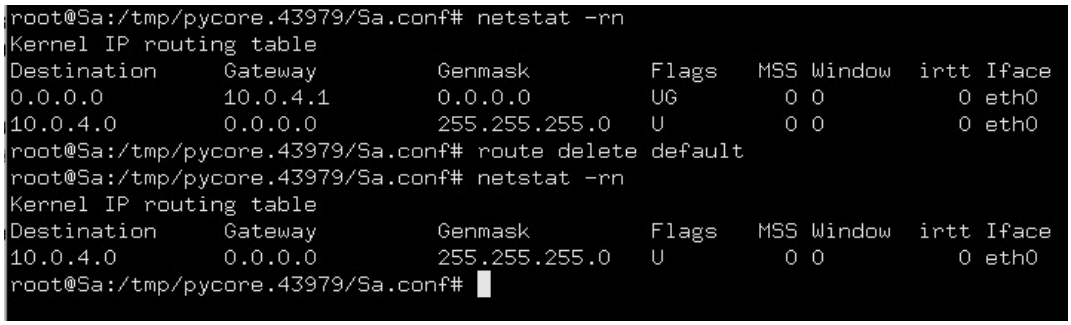



Figura 28: Comando *route delete* executado.

Pela Figura 28 conseguimos perceber que após retirada da rota por defeito da tabela de encaminhamento do servidor Sa, este apenas possui a entrada para o encaminhamento de dados que tenham como destino a sua própria sub-rede. Em consequência, e sabendo que a rota por defeito é a rota a seguir caso não exista uma entrada específica na tabela de encaminhamento para a rede destino requerida, ao retirarmos esta mesma entrada, estamos a impossibilitar a comunicação do servidor com dispositivos que se encontrem fora da sua sub-rede (10.0.4.0), pois este apesar de conseguir receber dados não irá conseguir responder. Assim, apresentamos de seguida o comando *ping* com origem num *host* de outra sub-rede (servidor Sc), provando que o servidor Sa não consegue comunicar de volta, pois a percentagem de pacotes perdidos (*packet loss*) é de 100%:

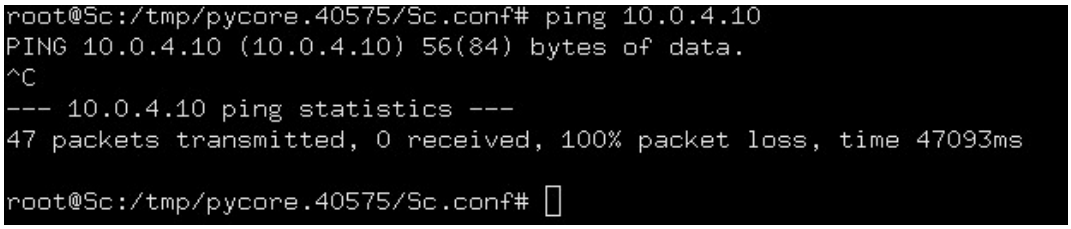



Figura 29: Comando *ping* ao servidor Sa.

**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.**

```
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.43979/Sa.conf#
```

Figura 30: Comandos utilizados para adicionar as rotas à tabela de encaminhamento.

Como podemos ver pela Figura 30, foram adicionadas estaticamente todas as sub-redes presentes na topologia LEI-RC utilizando como *gateway* o endereço IP do *router* Ra que anteriormente estava definida como rota por defeito, permitindo restaurar a conectividade para o servidor Sa.

**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.**

```
root@Sa:/tmp/pycore.43979/Sa.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=61 time=0.707 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=61 time=0.268 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=61 time=0.334 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=61 time=0.245 ms
^C
--- 10.0.6.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3079ms
rtt min/avg/max/mdev = 0.245/0.388/0.707/0.186 ms
root@Sa:/tmp/pycore.43979/Sa.conf#
```

Figura 31: Comando *ping* do servidor Sa para o servidor Sc.

Pela Figura 31, conseguimos denotar que o servidor está novamente acessível, uma vez que utilizamos o comando *ping* com destino no servidor Sc, estando este presente numa sub-rede diferente da do servidor Sa, provando assim que o servidor consegue novamente comunicar com os dispositivos localizados fora da sua sub-rede.

```
root@Sa:/tmp/pycore.43979/Sa.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.1.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.2.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.3.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
10.0.5.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.6.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.7.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.8.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
root@Sa:/tmp/pycore.43979/Sa.conf#
```

Figura 32: Tabela de Encaminhamento do servidor Sa.

De seguida, consultamos a nova tabela de encaminhamento do servidor (Figura 32), denotando as entradas que substituíram a rota por defeito, ou seja, entradas de todas as sub-redes da topologia. Adicionalmente, é de notar que a rota por defeito tem como objetivo, entre outros, reduzir a tabela de encaminhamento, agregando todas as rotas que possuem como endereço IP destino sub-redes exteriores, neste caso, ao servidor. Assim, o crescimento da tabela de encaminhamento após retirada da rota por defeito é notório como podemos comprovar pela comparação das Figuras 28 (após retirada da rota por defeito) e 32 (após introdução estática das rotas).

## 2.3 Questão 3 - Definição de Sub-redes

**a. Considere que dispõe apenas do endereço da rede IP 192.168.XXX.128/25, em que XXX é o decimal correspondendo ao seu número de grupo (PLXXX). 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.**

**192.168.135.128 /25**

11000000.10101000.100 00111. 10000000

25 bits de máscara de rede | 7 bits sub-redes

000 0000

**Cálculo das Sub-Redes:**

Como no enunciado é referido que o número de departamentos pode vir a aumentar em curto prazo, e como para quatro departamentos apenas precisaríamos de 2 bits, decidimos reservar 3 bits para a sub-rede, permitindo assim a expansão futura.
Desta forma, escolhemos os seguintes prefixos:

**000 ->** Departamento A
**001 ->** Departamento B
**010 ->** Departamento C
**011 ->** Departamento D

100
101
110
111 — Extras

**Cálculo dos Hosts:**

De acordo com o número de bits disponíveis para atribuição de endereços aos hosts (4 bits), atribuímos os mesmos de forma arbitrária apenas obedecendo à regra de incremento de um valor. Assim, obtivemos as seguintes sequências, atendendo ao facto de os valores com os bits todos a 0 (neste caso 0000) e a 1 (neste caso 1111) estão reservados, daí não serem utilizados:

**0001 ->** Host 1
**0010 ->** Host 2
**0011 ->** Host 3
**0100 ->** Host 4
...

**Endereços IP atribuídos:**

**Departamento A** = 192.168.135.128/28
- **RouterA** : 192.168.135.129/28
- **Sa** : 192.168.135.130/28
- **Monstro** : 192.168.135.131/28
- **Bela** : 192.168.135.132/28

**Departamento B** = 192.168.135.144/28
- **RouterB**: 192.168.135.145/28
- **Sb**: 192.168.135.146/28
- **Alladin** : 192.168.135.147/28
- **Jasmine**: 192.168.135.148/28

**Departamento C** = 192.168.135.160/28
- **RouterC**: 192.168.135.161/28
- **Sc** : 192.168.135.162/28
- **Eric** : 192.168.135.163/28
- **Ariel** : 192.168.135.164/28

**Departamento D** = 192.168.135.176/28
- **RouterD** : 192.168.135.177/28
- **Sc** : 192.168.135.178/28
- **Simba** : 192.168.135.179/28
- **Nala** : 192.168.135.180/28

**NOTA:** Por simplificação, na figura apresentada de seguida, os endereços IP das interfaces de ligação dos *routers* não estão presentes, mas é subentendida a sua presença. Assim, apresentamos uma vista simplificada da topologia com as diversas sub-redes (Departamentos):

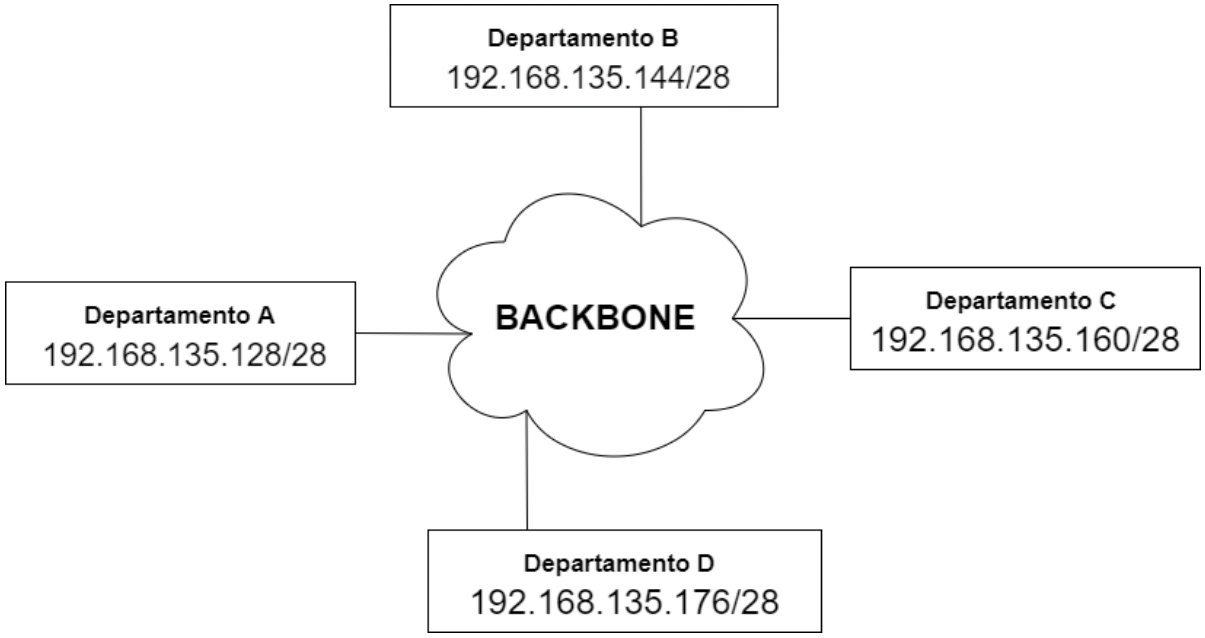



Figura 33: Vista simplificada da rede após aplicação de *Subnetting*.

**b. 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.**

A máscara de rede utilizada foi de /28 *bits* (255.255.255.240) uma vez que foram utilizados mais 3 *bits* para representar as diversas sub-redes. Assim, em cada departamento é possível interligar no máximo 14 *hosts* uma vez que todos têm disponíveis para o efeito 4 *bits* ($2^4$ - 2). Relativamente aos prefixos de sub-rede disponíveis para uso futuro, sobraram 4 prefixos possíveis, pois, uma vez que foram utilizados 3 *bits* para atribuição de sub-redes, há um total de oito combinações tendo quatro destas sido já atribuídas aos vários departamentos. Assim, sobraram os seguintes prefixos: 100, 101, 110 e 111.

**c. 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.**

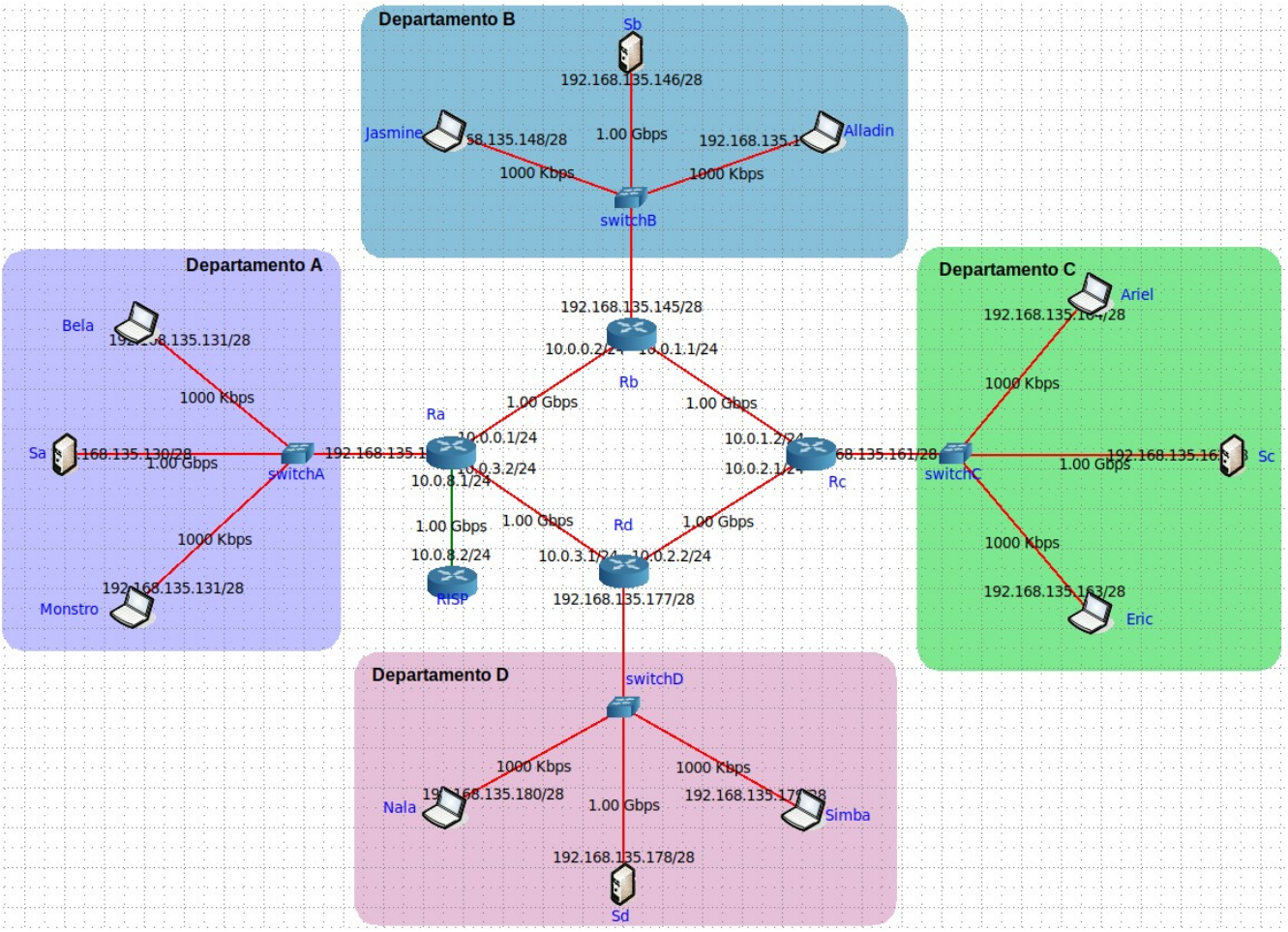



Figura 34: Topologia após aplicação de *Subnetting*.

Depois da modificação da topologia para a inclusão dos endereços obtidos por *subnetting*, de forma a testar a conectividade na topologia LEI-RC, seguimos o método utilizado na questão 1 da parte II presente neste mesmo relatório, isto é, começamos por verificar a **conectividade interna** nos vários departamentos, terminando na verificação da **conectividade inter-departamentos**. Assim, apresentamos a sequência de comandos *ping* utilizados para o efeito:

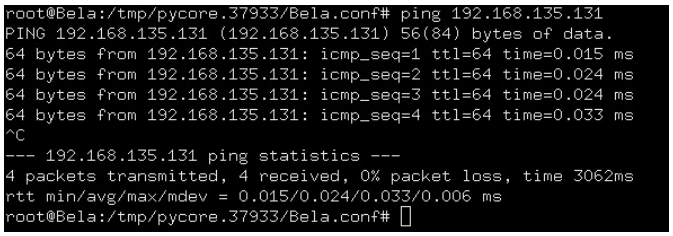



Figura 35: Departamento A (Bela - Monstro).

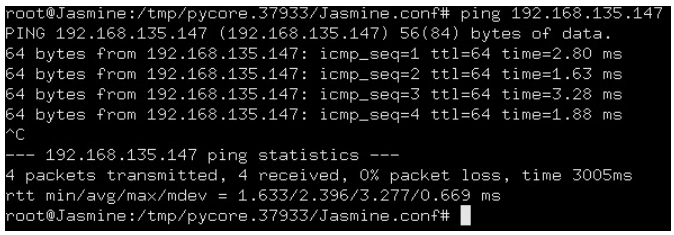



Figura 36: Departamento B (Jasmine - Alladin).


Figura 37: Departamento C (Ariel - Eric).

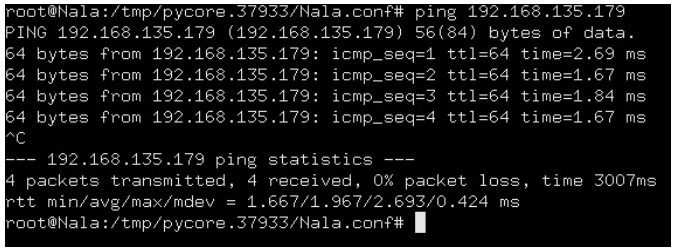



Figura 38: Departamento D (Nala - Simba).


Figura 39: Ping do Departamento A para o C.

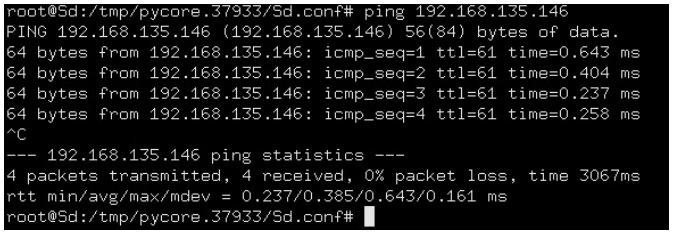



Figura 40: Ping do Departamento D para o B.

# 3 Conclusão

Com o finalizar deste trabalho, encontramo-nos, em geral, satisfeitos com o trabalho desenvolvido, tendo sido alcançados todos os objetivos propostos pelos docentes presentes nos enunciados da primeira e segunda partes que englobaram este segundo trabalho prático da unidade curricular.

Na **primeira parte** deste trabalho, foi-nos proposto a resolução de vários exercícios e problemas da área de redes, nomeadamente o desenvolvimento de uma topologia, captura de pacotes através da ferramenta *wireshark* e, por fim, a análise e manipulação de datagramas IP. Este desenvolvimento ocorreu sem grandes problemas estando o grupo bastante satisfeito com os resultados.

Na **segunda parte**, especificamos ainda mais o objeto de estudo dos exercícios, focando-nos nas temáticas de endereçamento, encaminhamento e técnicas de *subnetting*. Para além disto, foi necessário construir uma nova topologia que englobasse mais variáveis sendo, por isso, mais complexa que a desenvolvida anteriormente. Nesta parte conseguimos perceber que algumas alíneas poderiam ter sido resolvidas de maneira diferente, como, por exemplo, a alínea de inserção de novas rotas de forma estática no *router* Ra após a remoção da rota por defeito. Esta poderia ter sido resolvida de forma mais eficiente utilizando a técnica de *supernettting* diminuindo bastante o tamanho da tabela de encaminhamento do mesmo, no entanto, adicionamos entradas por cada sub-rede presente na topologia.

Em suma, através deste projeto, conseguimos compreender a difícil e necessária organização dos vários componentes da rede e a importância de cada um destes para o bom funcionamento da mesma. Para além disto, a construção e desenvolvimento deste trabalho prático permitiu a todo o grupo aprofundar os seus conhecimentos, mesmo que introdutórios, na área de redes. Em consequência, depois deste trabalho atingimos uma ampla percepção de alguns assuntos da área como encaminhamento, endereçamento, *subnetting* e fragmentação de datagramas.