

Universidade do Minho
Mestrado Integrado em Engenharia Informática

Redes de Computadores

TP2: Protocolo IPv4

Henrique Paz (a84372), Joana Afonso Gomes (A84912), João Neves (a81366)

October 2019

1 Parte I

Parte I

Pergunta 1

- (a) Active o *wireshark* ou o *tcpdump* no pc s1. Numa shell de s1, execute o comando *traceroute -I* para o endereço IP do host h5.

(Nós usamos o *wireshark*).

```
root@s1:/tmp/pycore.41800/s1.conf# traceroute -I 10.0.3.20
traceroute to 10.0.3.20 (10.0.3.20), 30 hops max, 60 byte packets
 1 10 (10.0.0.1) 0.032 ms 0.007 ms 0.006 ms
 2 10.0.1.2 (10.0.1.2) 0.020 ms 0.011 ms 0.011 ms
 3 * * *
 4 10.0.3.20 (10.0.3.20) 0.023 ms 0.020 ms 0.020 ms
```

Figura 1: Comando *traceroute -I* para o endereço IP do host h5

- (b) Registe e analise o tráfego ICMP enviado por s1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

No. Time	Source	Destination	Protocol	Length	Info
1 0.000000	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=1/256, ttl=1
2 0.000025	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
3 0.000031	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=2/512, ttl=1
4 0.000036	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
5 0.000041	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=3/768, ttl=1
6 0.000046	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
7 0.000050	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=4/1024, ttl=2
8 0.000068	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
9 0.000073	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=5/1280, ttl=2
10 0.000083	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
11 0.000087	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=6/1536, ttl=2
12 0.000095	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
13 0.000100	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=7/1792, ttl=3
14 0.000116	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=8/2048, ttl=3
15 0.000125	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=9/2304, ttl=3
16 0.000134	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=10/2560, ttl=4
17 0.000173	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0054, seq=10/2560, ttl=61
18 0.000178	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=11/2816, ttl=4
19 0.000195	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0054, seq=11/2816, ttl=61
20 0.000199	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=12/3072, ttl=4
21 0.000236	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0054, seq=12/3072, ttl=61
22 0.000242	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=13/3328, ttl=5
23 0.000257	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0054, seq=13/3328, ttl=61
24 0.000261	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=14/3584, ttl=5
25 0.000276	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0054, seq=14/3584, ttl=61
26 0.000280	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=15/3840, ttl=5
27 0.000327	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0054, seq=15/3840, ttl=61
28 0.000335	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=16/4096, ttl=6
30 0.000527	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x0054, seq=17/4352, ttl=6
31 0.000546	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0054, seq=17/4352, ttl=61
32 5.012223	00:00:00_aa:00:02	00:00:00_aa:00:03	ARP	42	Who has 10.0.0.1? Tell 10.0.0.10
33 5.012272	00:00:00_aa:00:03	00:00:00_aa:00:02	ARP	42	Who has 10.0.0.10? Tell 10.0.0.1
34 5.012323	00:00:00_aa:00:03	00:00:00_aa:00:02	ARP	42	10.0.0.1 is at 00:00:00_aa:00:03
35 5.012305	00:00:00_aa:00:02	00:00:00_aa:00:03	ARP	42	10.0.0.10 is at 00:00:00_aa:00:02
36 5.631347	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
37 5.631428	fe80::200:ff:feaa3:ff02::5	::5	OSPF	90	Hello Packet

Figura 2: Tráfego ICMP enviado e recebido por s1.

Nós observamos que enquanto o TTL é menor ou igual a 3, não obtemos nenhuma resposta.

Isto coincide com o comportamento esperado, dado que enquanto o TTL não for 4, o pacote não pode ser enviado (como será analisado na resposta seguinte).

- (c) **Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino h5? Verifique na prática que a sua resposta está correta.**

O valor mínimo do TTL para alcançar o destino h5 é de 4.

- (d) **Qual o valor médio do tempo de ida-e-volta (*Round Time Trip*) obtido?**

Visto que o tempo apresentado na coluna "*Time*" do Wireshark está com a opção de "Seconds Since Previous Captured Packet" (segundo desde o último pacote capturado). Assim, para calcular o RTT, basta fazer uma média de todos os tempos das linhas de respostas. $(0.000173s + 0.000195s + 0.000236s + 0.000257s + 0.000276s + 0.000327 + 0.000357s + 0.000546s) / 8 \approx 0.000296s = 0.296ms$.

Pergunta 2

- (a) **Qual é o endereço IP da interface ativa do seu computador?**

O IP é **192.168.100.156**.

```


▼ Internet Protocol Version 4, Src: 192.168.100.156, Dst: 193.136.9.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ► Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 512
    Identification: 0x5d1e (23838)
  ▼ Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. ... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
  ► Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x6a14 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.100.156
    Destination: 193.136.9.254

```

Figura 3.

- (b) Qual é o valor do campo protocolo? O que identifica?

O protocolo é o **ICMP** (*Internet Control Message Protocol*). Este é usado para apresentar mensagens de reporte de erros e mostrar o estatuto de pedido ou resposta.



No.	Time	Source	Destination	Protocol	Length
4	1.915562256	192.168.100.156	193.136.9.254	ICMP	526
▼ Internet Control Message Protocol					
Type: 8 (Echo (ping) request)					
Code: 0					
Checksum: 0xe6a8 [correct]					
[Checksum Status: Good]					

Figura 4.

- (c) Quantos bytes tem o cabeçalho IP(v4)? Quantos bytes tem o campo de dados (*payload*) do datagrama? Como se calcula o tamanho do *payload*?

O cabeçalho IPv4 tem 20 *bytes*. Foram enviados 512 bytes, sendo portanto o *payload* calculado subtraindo os 20 bytes do cabeçalho do IP ao número de *bytes* enviados.

```

▼ Internet Protocol Version 4, Src: 192.168.100.156, Dst: 193.136.9.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ► Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 512
    Identification: 0x5d1d (23837)
  ▼ Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set

```

Figura 5.

- (d) O datagrama IP foi fragmentado? Justifique.

Não foi fragmentado, dado que o *Fragment offset* está a 0.

```

▼ Internet Protocol Version 4, Src: 192.168.100.156, Dst: 193.136.9.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ► Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 512
    Identification: 0x5d1d (23837)
  ▼ Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0

```

Figura 6.

- (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.

Os campos do cabeçalho IP que variam de pacote para pacote são o TTL (*Time to live*) e a *Identification*.

Exemplo:

```
▼ Internet Protocol Version 4, Src: 192.168.100.156, Dst: 193.136.9.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ► Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 512
    Identification: 0x5d1d (23837) ←
  ▼ Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
  ► Time to live: 1 ←
    Protocol: ICMP (1)
    Header checksum: 0x6a15 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.100.156
    Destination: 193.136.9.254
```

Figura 7.

```
▼ Internet Protocol Version 4, Src: 192.168.100.254, Dst: 192.168.100.156
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ► Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 540
    Identification: 0x2598 (9624) ←
  ▼ Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0
  Time to live: 64 ←
    Protocol: ICMP (1)
    Header checksum: 0x079e [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.100.254
    Destination: 192.168.100.156
```

Figura 8.

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

A Identificação aumenta 1 por pacote e o TTL aumenta 1 por cada 3 pacotes.

Exemplo da diferença da identificação:

```

▶ Frame 4: 526 bytes on wire (4208 bits), 526 bytes captured (4208 bits) on interface 0
▼ Ethernet II, Src: AsustekC_35:e0:34 (2c:4d:54:35:e0:34), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
  ▶ Destination: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
  ▶ Source: AsustekC_35:e0:34 (2c:4d:54:35:e0:34)
  Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 192.168.100.156, Dst: 193.136.9.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 512
  Identification: 0x5d1d (23837)
  ▼ Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0

```

Figura 9.

```

▶ Frame 5: 526 bytes on wire (4208 bits), 526 bytes captured (4208 bits) on interface 0
▼ Ethernet II, Src: AsustekC_35:e0:34 (2c:4d:54:35:e0:34), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
  ▶ Destination: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
  ▶ Source: AsustekC_35:e0:34 (2c:4d:54:35:e0:34)
  Type: IPv4 (0x0800)
▼ Internet Protocol Version 4, Src: 192.168.100.156, Dst: 193.136.9.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 512
  Identification: 0x5d1e (23838)
  ▼ Flags: 0x0000
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    ...0 0000 0000 0000 = Fragment offset: 0

```

Figura 10.

No.	Info
4	Echo (ping) request id=0x54a8, seq=1/256, ttl=1 (no response found!)
5	Echo (ping) request id=0x54a8, seq=2/512, ttl=1 (no response found!)
6	Echo (ping) request id=0x54a8, seq=3/768, ttl=1 (no response found!)
7	Echo (ping) request id=0x54a8, seq=4/1024, ttl=2 (reply in 26)
8	Echo (ping) request id=0x54a8, seq=5/1280, ttl=2 (reply in 27)
9	Echo (ping) request id=0x54a8, seq=6/1536, ttl=2 (reply in 28)
10	Echo (ping) request id=0x54a8, seq=7/1792, ttl=3 (reply in 29)
11	Echo (ping) request id=0x54a8, seq=8/2048, ttl=3 (reply in 30)
12	Echo (ping) request id=0x54a8, seq=9/2304, ttl=3 (reply in 31)
13	Echo (ping) request id=0x54a8, seq=10/2560, ttl=4 (reply in 32)
14	Echo (ping) request id=0x54a8, seq=11/2816, ttl=4 (reply in 33)
15	Echo (ping) request id=0x54a8, seq=12/3072, ttl=4 (reply in 34)
16	Echo (ping) request id=0x54a8, seq=13/3328, ttl=5 (reply in 35)
17	Echo (ping) request id=0x54a8, seq=14/3584, ttl=5 (reply in 36)
18	Echo (ping) request id=0x54a8, seq=15/3840, ttl=5 (reply in 37)
19	Echo (ping) request id=0x54a8, seq=16/4096, ttl=6 (reply in 38)
23	Echo (ping) request id=0x54a8, seq=17/4352, ttl=6 (reply in 39)
24	Echo (ping) request id=0x54a8, seq=18/4608, ttl=6 (reply in 40)

Figura 11: Exemplo do aumento do TTL.

- (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?

O valor do campo TTL é 254 e permanece inalterado para todas as mensagens de resposta.

No.	Time	Source	Destination	Protocol	Length	Info
20	1.916190745	192.168.100.254	192.168.100.156	ICMP	554	Time-to-live exceeded (Time to live exceeded in transit)
21	1.916198689	192.168.100.254	192.168.100.156	ICMP	554	Time-to-live exceeded (Time to live exceeded in transit)
22	1.916200261	192.168.100.254	192.168.100.156	ICMP	554	Time-to-live exceeded (Time to live exceeded in transit)
26	1.918028495	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=4/1024, ttl=254 (request in 7)
27	1.918570472	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=5/1280, ttl=254 (request in 8)
28	1.919075022	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=6/1536, ttl=254 (request in 9)
29	1.919506851	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=7/1792, ttl=254 (request in 10)
30	1.920040344	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=8/2048, ttl=254 (request in 11)
31	1.920665606	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=9/2304, ttl=254 (request in 12)
32	1.921152939	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=10/2560, ttl=254 (request in 13)
33	1.921661925	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=11/2816, ttl=254 (request in 14)
34	1.922238459	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=12/3072, ttl=254 (request in 15)
35	1.922762536	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=13/3328, ttl=254 (request in 16)
36	1.923258071	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=14/3584, ttl=254 (request in 17)
37	1.923758311	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=15/3840, ttl=254 (request in 18)
38	1.924266225	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=16/4096, ttl=254 (request in 19)
39	1.924855540	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=17/4352, ttl=254 (request in 23)
40	1.925347373	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=18/4608, ttl=254 (request in 24)
41	1.925855738	193.136.9.254	192.168.100.156	ICMP	526	Echo (ping) reply id=0x54a8, seq=19/4864, ttl=254 (request in 25)
4	1.915562256	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=1/256, ttl=1 (no response found!)
5	1.915569959	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=2/512, ttl=1 (no response found!)
6	1.915573158	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=3/768, ttl=1 (no response found!)
7	1.915576949	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=4/1024, ttl=2 (reply in 26)
8	1.915580047	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=5/1280, ttl=2 (reply in 27)
9	1.915583256	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=6/1536, ttl=2 (reply in 28)
10	1.915587056	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=7/1792, ttl=3 (reply in 29)
11	1.915602692	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=8/2048, ttl=3 (reply in 30)
12	1.915605760	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=9/2304, ttl=3 (reply in 31)
13	1.915609401	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=10/2560, ttl=4 (reply in 32)
14	1.915612194	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=11/2816, ttl=4 (reply in 33)
15	1.915615218	192.168.100.156	193.136.9.254	ICMP	526	Echo (ping) request id=0x54a8, seq=12/3072, ttl=4 (reply in 34)

Figura 12: Tráfego capturado ordenado por endereço destino.

Pergunta 3

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

Houve necessidade de fragmentar o pacote inicial porque o seu tamanho inicial, 4207, é demasiado grande para circular na rede

- (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?

Sabemos que o segmento foi fragmentado porque a *flag More Fragments* está a 1. O que indica que se trata do primeiro fragmento é a *flag Fragment offset* estar a 0. O tamanho do datagrama IP é 1500 bytes.


```

▶ Frame 2: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
▶ Ethernet II, Src: AsustekC_35:e0:34 (2c:4d:54:35:e0:34), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)
▼ Internet Protocol Version 4, Src: 192.168.100.156, Dst: 193.136.9.254
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0xbf13 (48915)
    ▼ Flags: 0x2000, More fragments
        0... .. = Reserved bit: Not set
        .0.. .. = Don't fragment: Not set
        ..1. .... = More fragments: Set
        ...0 0000 0000 0000 = Fragment offset: 0
    ▶ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0xe442 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.100.156
    Destination: 193.136.9.254
    Reassembled IPv4 in frame: 4

```

Figura 13.

- (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? Sabemos que não se trata do primeiro fragmento porque a **Fragment Offset** é diferente de 0. Há mais fragmentos porque o a **More Fragments** está a 1.

```

▼ Flags: 0x20b9, More fragments
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ➡ ..1. .... = More fragments: Set
    ...0 0000 1011 1001 = Fragment offset: 185

```

Figura 14.

- (d) Quantos fragmentos foram criados a partir do datagrama original? Como se detecta o último fragmento correspondente ao datagrama original?

Foram criados 3 fragmentos a partir do datagrama original, visto que o a Identification muda ao fim de 3 *frames*. Detetamos que é o último fragmento porque a *ag More fragments* se encontra a 0.

```

▼ Flags: 0x0172
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ➡ ..0. .... = More fragments: Not set
    ...0 0001 0111 0010 = Fragment offset: 370

```

Figura 15.

- (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.

Nota: O *Fragment offset* aparece nos *prints* em blocos de *bytes* em vez de *bytes*.

Parte II

Pergunta 1

Atenda aos endereços IP atribuídos automaticamente pelo CORE aos diversos equipamentos da topologia.

- (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.

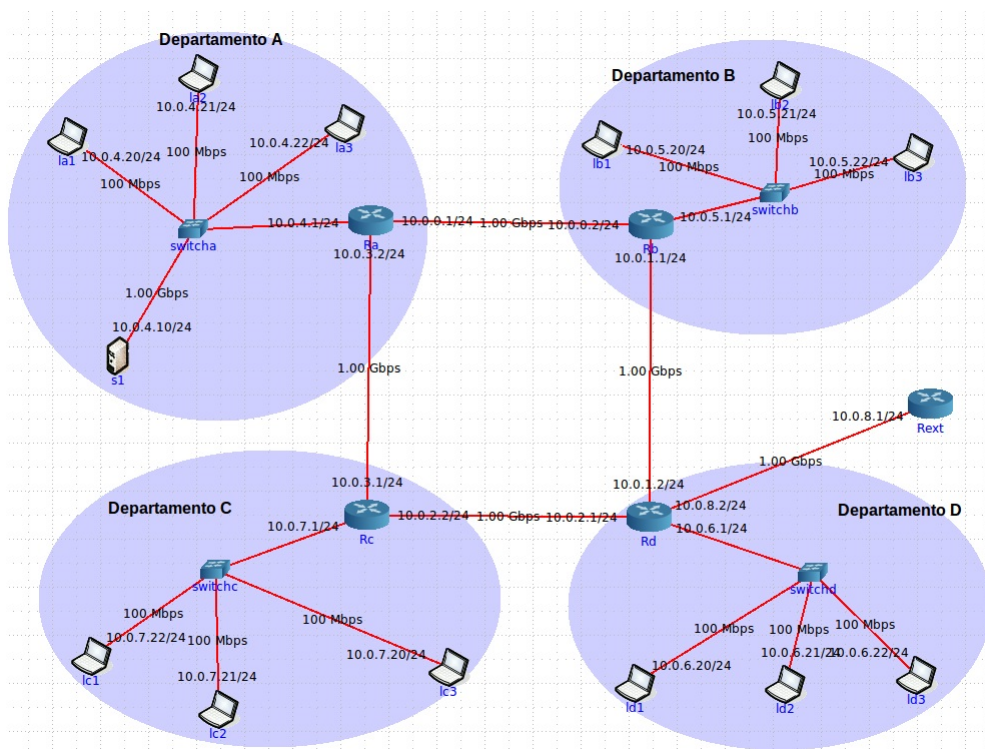


Figura 16: Topologia da rede.

- (b) Trata-se de endereços públicos ou privados? Porquê?

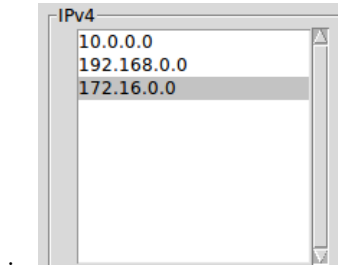


Figura 17: IP *addresses*.

Pela análise de *IP addresses*, e sabendo que a máscara de rede é sempre 255.0.0.0, concluímos que os intervalos dos endereços privados são:

10.0.0.0 — 10.255.0.0
192.168.0.0 — 192.168.255.255
172.16.0.0 — 172.16.255.255

Analisando os IPs dos *laptops*, como por exemplo, o ip de la1, 10.0.4.20/**24**, observamos que a máscara tem **24** bytes (notação CIDR), ou seja, a máscara de rede é **255.255.255.0** (notação decimal).

- (c) Por que razão não é atribuído um endereço IP aos *switches*?

Os *switch* servem para fazer a interligação de equipamentos. Acessando o endereço MAC de cada um dos dispositivos que a ele estão ligados, conseguem redirecionar pacotes entre eles.

Sendo os **switches** da segunda camada de ligação, são transparentes à camada de ligação 3 onde estão os endereços de IP, daí não ser preciso atribuir-lhes um endereço IP.

- (d) Usando o comando `ping` certifique-se que existe conectividade IP entre os *laptops* dos vários departamentos e o servidor do departamento A (basta certificar-se da conectividade de um *laptop* por departamento).

Usamos os *laptops* la1, lb1, lc1, ld1, respectivamente dos departamentos A, B, C e D, para nos certificarmos da existência de conectividade entre os *laptops* e o servidor.

```
root@la1:/tmp/pycore.54006/la1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=64 time=0.100 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=64 time=0.063 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=64 time=0.022 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=64 time=0.070 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=64 time=0.063 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=64 time=0.068 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4998ms
rtt min/avg/max/mdev = 0.022/0.064/0.100/0.023 ms
root@la1:/tmp/pycore.54006/la1.conf# █
```

Figura 18: Conectividade entre o *laptop* la1 e o servidor s1.

```
root@lb1:/tmp/pycore.54006/lb1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=62 time=0.099 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=62 time=0.099 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=62 time=0.102 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=62 time=0.104 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=62 time=0.112 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=62 time=0.115 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4998ms
rtt min/avg/max/mdev = 0.099/0.105/0.115/0.008 ms
root@lb1:/tmp/pycore.54006/lb1.conf# █
```

Figura 19: Conectividade entre o *laptop* lb1 e o servidor s1.

```

root@lc1:/tmp/pycore.54006/lc1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=62 time=0.078 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=62 time=0.193 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=62 time=0.092 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=62 time=0.093 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=62 time=0.095 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=62 time=0.101 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4996ms
rtt min/avg/max/mdev = 0.078/0.108/0.193/0.040 ms
root@lc1:/tmp/pycore.54006/lc1.conf# █

```

Figura 20: Conectividade entre o *laptop* lc1 e o servidor s1.

```

root@ld1:/tmp/pycore.54006/ld1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=61 time=0.097 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=61 time=0.116 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=61 time=0.151 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=61 time=0.116 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=61 time=0.121 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=61 time=0.119 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4996ms
rtt min/avg/max/mdev = 0.097/0.120/0.151/0.015 ms
root@ld1:/tmp/pycore.54006/ld1.conf# █

```

Figura 21: Conectividade entre o *laptop* ld1 e o servidor s1.

Como podemos observar pelas figuras acima, existe.

- (e) Verifique se existe conectividade IP do *router* de acesso Rext para o servidor s1.

```

root@Rext:/tmp/pycore.54006/Rext.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=61 time=0.123 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=61 time=0.135 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=61 time=0.151 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=61 time=0.153 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=61 time=0.138 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=61 time=0.158 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4997ms
rtt min/avg/max/mdev = 0.123/0.143/0.158/0.012 ms
root@Rext:/tmp/pycore.54006/Rext.conf# █

```

Figura 22: Conectividade entre o *router* de acesso Rext e o servidor s1.

Verificamos que existe conectividade.

Pergunta 2

Para o *router* e um *laptop* do departamento B:

- (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*).

```
root@Rb:/tmp/pycore.37136/Rb.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 10.0.1.2 255.255.255.0 UG 0 0 0 eth1
10.0.7.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.8.0 10.0.1.2 255.255.255.0 UG 0 0 0 eth1
root@Rb:/tmp/pycore.37136/Rb.conf#
```

Figura 23: Tabela de encaminhamento do *router* Rb.

```
root@lb1:/tmp/pycore.37136/lb1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.5.1 0.0.0.0 UG 0 0 0 eth0
10.0.5.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@lb1:/tmp/pycore.37136/lb1.conf#
```

Figura 24: Tabela de encaminhamento do *laptop* lb1.

Analisando o output e confrontando com manuais do *netstat*, concluímos o seguinte quanto às entradas da tabela:

- **Destination** indica a (sub)rede de destino.
- **Gateway** mostra-nos a *gateway* (“porta de entrada”).
- **Genmask** apresenta a máscara de rede.
- **Flags** exibe as *flags* que descrevem a rota. Neste caso apenas há **U** e **UG**, sendo que no primeiro caso significa que a interface a ser usada está ativa, e no segundo caso que para além disso a rota usa uma *gateway*.

Na tabela de encaminhamento do *router* **Rb** vemos que:

- Pacotes que tenham como destino um equipamento da sub-rede 10.0.2.0, 10.0.5.0, 10.0.6.0 ou 10.0.7.0 têm de passar pelo *router* **Rd** (*gateway* 10.0.1.2)

- Pacotes que tenham como destino um equipamento da sub-rede 10.0.3.0, 10.0.4.0 e 10.0.7.0 têm de passar pelo router **Ra** (*gateway* 10.0.0.1)
- Pacotes com outros destinos, como a *gateway* é 0.0.0.0, podem ir por qualquer rota

Tendo agora em conta a tabela da figura (...) (*laptop lb1*):

- Analisando a primeira linha verificamos que há a opção de que, para qualquer que seja o endereço de destino do pacote, este passa pelo *router Rb* (*gateway* 10.0.5.1).
- Na segunda linha vemos a outra hipótese que é o destino do pacote ser a sub-rede do Departamento B, podendo optar por uma das rotas.

(b) **Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema).**

Usamos o comando *ps ax* para saber quais os processos que estão a decorrer. Em semelhança à alínea anterior, analisamos para um *router* (**Rb**) e para um *laptop* no departamento desse *router* (**lb1**).

```
root@Rb:/tmp/pycore.49574/Rb.conf# ps ax
PID TTY      STAT   TIME COMMAND
  1 ?        S      0:00 /usr/sbin/vnoded -v -c /tmp/pycore.49574/Rb -l /tmp/p
 48 ?        Ss     0:00 /usr/lib/quagga/zebra -u root -g root -d
 71 ?        Ss     0:00 /usr/lib/quagga/ospfd -u root -g root -d
 72 ?        Ss     0:00 /usr/lib/quagga/ospf6d -u root -g root -d
 77 pts/12   Ss     0:00 /bin/bash
131 pts/12   R+     0:00 ps ax
root@Rb:/tmp/pycore.49574/Rb.conf#
```

Figura 25: Processos a decorrer no *router* Rb.

```
root@lb1:/tmp/pycore.49574/lb1.conf# ps ax
PID TTY      STAT   TIME COMMAND
  1 ?        S      0:00 /usr/sbin/vnoded -v -c /tmp/pycore.49574/lb1 -l /tmp/p
 73 pts/12   Ss     0:00 /bin/bash
127 pts/12   R+     0:00 ps ax
root@lb1:/tmp/pycore.49574/lb1.conf#
```

Figura 26: Processos a decorrer no *laptop* lb1.

No caso do *router Rb* constata-se na coluna **COMMAND** que há um protocolo usado, o **ospfd**. Através de pesquisa, concluímos que este protocolo (*Open Shortest Path First*) é usado para encontrar o melhor caminho entre *source* e destino. Deduzimos assim que o encaminhamento do router Ra é **dinâmico**.

Já no caso do *laptop lb1* concluímos ser **estático** dada a não utilização de protocolos.

- (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 s1 localizado no departamento A. Use o comando `route delete` para o efeito. Que implicação tem esta medida para os utilizadores da empresa que acedem ao servidor? Justifique.

Após executarmos `route delete` da rota por defeito corremos para o servidor s1 o previamente já usado comando `netstat` e verificamos que, como seria suposto, essa rota saiu da sua tabela de encaminhamento.

```
root@s1:/tmp/pycore.37136/s1.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@s1:/tmp/pycore.37136/s1.conf# route delete default
root@s1:/tmp/pycore.37136/s1.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.4.0         0.0.0.0         255.255.255.0   U        0  0          0 eth0
root@s1:/tmp/pycore.37136/s1.conf#
```

Figura 27: Tabela de encaminhamento do servidor s1 antes e depois de remover a rota `default`.

De seguida, usamos o comando `ping` para testar a conectividade entre *laptops* dos diferentes departamentos com o servidor s1.

```
root@la1:/tmp/pycore.37136/la1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data:
64 bytes from 10.0.4.10: icmp_req=1 ttl=64 time=0.061 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=64 time=0.074 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=64 time=0.078 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=64 time=0.092 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=64 time=0.075 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=64 time=0.078 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4998ms
rtt min/avg/max/mdev = 0.061/0.076/0.092/0.011 ms
root@la1:/tmp/pycore.37136/la1.conf#
```

Figura 28: Conectividade entre o *laptop* la1 e o servidor s1.

```
root@lb1:/tmp/pycore.37136/lb1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
^C
--- 10.0.4.10 ping statistics ---
15 packets transmitted, 0 received, 100% packet loss, time 14113ms

root@lb1:/tmp/pycore.37136/lb1.conf# █
```

Figura 28: Conectividade entre o *laptop* lb1 e o servidor s1.

```
root@lc1:/tmp/pycore.37136/lc1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
^C
--- 10.0.4.10 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2016ms

root@lc1:/tmp/pycore.37136/lc1.conf# █
```

Figura 29: Conectividade entre o *laptop* lc1 e o servidor s1.

```
root@ld1:/tmp/pycore.37136/ld1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
^C
--- 10.0.4.10 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1009ms

root@ld1:/tmp/pycore.37136/ld1.conf# █
```

Figura 30: Conectividade entre o *laptop* ld1 e o servidor s1.

Concluimos que só o *laptop* do Departamento A tem conectividade com o servidor s1. Nos outros *laptops* são enviados pacotes mas nenhum é recebido, levando-nos a concluir que estes não têm conectividade com o servidor.

- (d) Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor s1 por forma a contornar a restrição imposta na alínea c). Utilize para o efeito o comando `route add` e registre os comandos que usou.

```

root@s1:/tmp/pycore.54985/s1.conf# route delete default
root@s1:/tmp/pycore.54985/s1.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.4.0        0.0.0.0         255.255.255.0   U        0 0          0 eth0
root@s1:/tmp/pycore.54985/s1.conf# route add -net 10.0.5.0 netmask 255.255.255.0
gw 10.0.4.1
root@s1:/tmp/pycore.54985/s1.conf# route add -net 10.0.6.0 netmask 255.255.255.0
gw 10.0.4.1
root@s1:/tmp/pycore.54985/s1.conf# route add -net 10.0.7.0 netmask 255.255.255.0
gw 10.0.4.1
root@s1:/tmp/pycore.54985/s1.conf# netstat -rn
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
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
root@s1:/tmp/pycore.54985/s1.conf# █

```

Figura 31: Adição de rotas estáticas para restaurar a conectividade.

Exemplificando com o primeiro comando usado, este tem como objetivo fazer com que pacotes com máscara 24 (255.255.255.0) que tenham como destino a sub-rede 10.0.5.0 entrem necessariamente na *gateway* 10.0.4.1. Repetimos o mesmo processo para os pacotes que tenham o destino 10.0.3.0.

- (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@la1:/tmp/pycore.37136/la1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data:
64 bytes from 10.0.4.10: icmp_req=1 ttl=64 time=0.061 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=64 time=0.037 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=64 time=0.069 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=64 time=0.061 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=64 time=0.038 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=64 time=0.077 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4997ms
rtt min/avg/max/mdev = 0.037/0.057/0.077/0.015 ms
root@la1:/tmp/pycore.37136/la1.conf# █

```

Figura 32: Conectividade entre o *laptop* la1 e o servidor s1.

```

root@lb1:/tmp/pycore.37171/lb1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=62 time=0.047 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=62 time=0.118 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=62 time=0.123 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=62 time=0.112 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=62 time=0.118 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=62 time=0.132 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4996ms
rtt min/avg/max/mdev = 0.047/0.108/0.132/0.029 ms
root@lb1:/tmp/pycore.37171/lb1.conf# █

```

Figura 33: Conectividade entre o *laptop* lb1 e o servidor s1.

```

root@lc1:/tmp/pycore.37171/lc1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=62 time=0.099 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=62 time=0.155 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=62 time=0.457 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=62 time=0.109 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=62 time=0.122 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=62 time=0.124 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.099/0.177/0.457/0.127 ms
root@lc1:/tmp/pycore.37171/lc1.conf# █

```

Figura 34: Conectividade entre o *laptop* lc1 e o servidor s1.

```

root@ld1:/tmp/pycore.37171/ld1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_req=1 ttl=61 time=0.181 ms
64 bytes from 10.0.4.10: icmp_req=2 ttl=61 time=0.146 ms
64 bytes from 10.0.4.10: icmp_req=3 ttl=61 time=0.207 ms
64 bytes from 10.0.4.10: icmp_req=4 ttl=61 time=0.149 ms
64 bytes from 10.0.4.10: icmp_req=5 ttl=61 time=0.151 ms
64 bytes from 10.0.4.10: icmp_req=6 ttl=61 time=0.146 ms
^C
--- 10.0.4.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.146/0.163/0.207/0.025 ms
root@ld1:/tmp/pycore.37171/ld1.conf# █

```

Figura 35: Conectividade entre o *laptop* ld1 e o servidor s1.

Pergunta 3

- (a) Considere que dispõe apenas do endereço de rede IP $172.yyx.32.0/20$, em que “yy” são os dígitos correspondendo ao seu número de grupo (Gyy) e “x” é o dígito correspondente ao seu turno prático (PLx). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo a rede de acesso e core inalteradas) e atribua endereços às interfaces dos vários sistemas envolvidos. Deve justificar as opções usadas.

Tendo em conta o endereço $172.071.32.0/20$: Como temos 4 departamentos, decidimos usar 3 bits para o *host id*. Existem $2^3 - 2 = 6$ endereços disponíveis (retiram-se 2 bits porque um está reservado para o *default* (000) e outro para o *broadcast* (111).

$172.071.00100000.0/20$

(8 bits) (8 bits) rede id host id (8 bits)

Figura 36: Explicação do novo endereçamento.

Como falta um bit ainda nos 8 bits Como adicionamos 3 bits, a máscara de rede fica agora $255.255.254.0 (/23)$.

- Ao Departamento A é atribuído o endereço $172.071.34.0/23$.
- Ao Departamento B é atribuído o endereço $172.071.36.0/23$.
- Ao Departamento C é atribuído o endereço $172.071.38.0/23$.
- Ao Departamento D é atribuído o endereço $172.071.40.0/23$.

Por exemplo, no caso do Departamento A, o *router* terá o endereço IP $172.071.34.1/23$ e o servidor $172.071.34.10/23$.

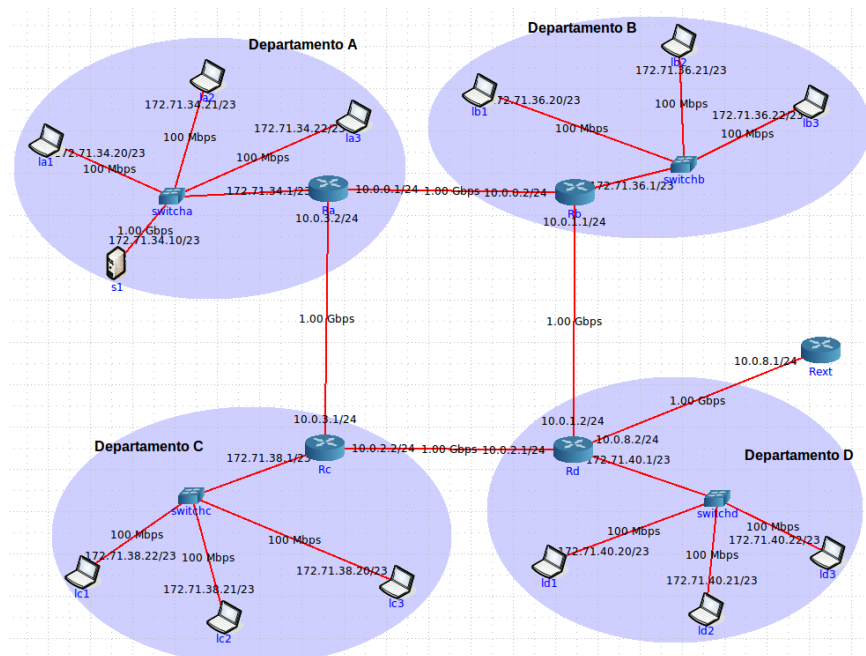


Figura 37: Topologia de rede com o novo endereçamento.

- (b) Qual a máscara de rede que usou (em notação decimal)? Quantos interfaces IP pode interligar em cada departamento? Justifique.

Usamos a máscara de rede 255.255.254.0 (em notação CIDR, /23). Em cada departamento podem-se interligar $2^9 - 2 = 510$ interfaces IP.

- (c) Garanta e verifique que a conectividade IP entre as várias redes locais da organização MIEI-RC é mantida. Explique como procedeu.

Testamos a conectividade (com o comando *ping*) entre *laptops* dos diversos departamentos (la1, lb1, lc1, ld1) e o servidor s1 e entre *laptops* de departamentos diferentes.

```

root@la1:/tmp/pycore.49574/la1.conf# ping 172.71.34.10
PING 172.71.34.10 (172.71.34.10) 56(84) bytes of data.
64 bytes from 172.71.34.10: icmp_req=1 ttl=64 time=0.055 ms
64 bytes from 172.71.34.10: icmp_req=2 ttl=64 time=0.032 ms
64 bytes from 172.71.34.10: icmp_req=3 ttl=64 time=0.071 ms
64 bytes from 172.71.34.10: icmp_req=4 ttl=64 time=0.027 ms
64 bytes from 172.71.34.10: icmp_req=5 ttl=64 time=0.071 ms
64 bytes from 172.71.34.10: icmp_req=6 ttl=64 time=0.072 ms
^C
--- 172.71.34.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4998ms
rtt min/avg/max/mdev = 0.027/0.054/0.072/0.020 ms
root@la1:/tmp/pycore.49574/la1.conf# █

```

Figura 38: Conectividade entre o *laptop* la1 e o servidor s1.

```

root@lb1:/tmp/pycore.49574/lb1.conf# ping 172.71.34.10
PING 172.71.34.10 (172.71.34.10) 56(84) bytes of data.
64 bytes from 172.71.34.10: icmp_req=1 ttl=62 time=0.145 ms
64 bytes from 172.71.34.10: icmp_req=2 ttl=62 time=0.075 ms
64 bytes from 172.71.34.10: icmp_req=3 ttl=62 time=0.062 ms
64 bytes from 172.71.34.10: icmp_req=4 ttl=62 time=0.056 ms
64 bytes from 172.71.34.10: icmp_req=5 ttl=62 time=0.076 ms
64 bytes from 172.71.34.10: icmp_req=6 ttl=62 time=0.070 ms
^C
--- 172.71.34.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.056/0.080/0.145/0.031 ms
root@lb1:/tmp/pycore.49574/lb1.conf# █

```

Figura 39: Conectividade entre o *laptop* lb1 e o servidor s1.

```

root@lc1:/tmp/pycore.49574/lc1.conf# ping 172.71.34.10
PING 172.71.34.10 (172.71.34.10) 56(84) bytes of data.
64 bytes from 172.71.34.10: icmp_req=1 ttl=62 time=0.112 ms
64 bytes from 172.71.34.10: icmp_req=2 ttl=62 time=0.051 ms
64 bytes from 172.71.34.10: icmp_req=3 ttl=62 time=0.059 ms
64 bytes from 172.71.34.10: icmp_req=4 ttl=62 time=0.055 ms
64 bytes from 172.71.34.10: icmp_req=5 ttl=62 time=0.066 ms
64 bytes from 172.71.34.10: icmp_req=6 ttl=62 time=0.066 ms
^C
--- 172.71.34.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.051/0.068/0.112/0.020 ms
root@lc1:/tmp/pycore.49574/lc1.conf# █

```

Figura 40: Conectividade entre o *laptop* lc1 e o servidor s1.

```

root@ld1:/tmp/pycore.49574/ld1.conf# ping 172.71.34.10
PING 172.71.34.10 (172.71.34.10) 56(84) bytes of data.
64 bytes from 172.71.34.10: icmp_req=1 ttl=61 time=0.086 ms
64 bytes from 172.71.34.10: icmp_req=2 ttl=61 time=0.169 ms
64 bytes from 172.71.34.10: icmp_req=3 ttl=61 time=0.136 ms
64 bytes from 172.71.34.10: icmp_req=4 ttl=61 time=0.135 ms
64 bytes from 172.71.34.10: icmp_req=5 ttl=61 time=0.075 ms
64 bytes from 172.71.34.10: icmp_req=6 ttl=61 time=0.124 ms
^C
--- 172.71.34.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.075/0.120/0.169/0.034 ms
root@ld1:/tmp/pycore.49574/ld1.conf# █

```

Figura 41: Conectividade entre o *laptop* ld1 e o servidor s1.

```

root@la1:/tmp/pycore.49574/la1.conf# ping 172.71.36.20
PING 172.71.36.20 (172.71.36.20) 56(84) bytes of data.
64 bytes from 172.71.36.20: icmp_req=1 ttl=62 time=0.088 ms
64 bytes from 172.71.36.20: icmp_req=2 ttl=62 time=0.167 ms
64 bytes from 172.71.36.20: icmp_req=3 ttl=62 time=0.044 ms
64 bytes from 172.71.36.20: icmp_req=4 ttl=62 time=0.064 ms
64 bytes from 172.71.36.20: icmp_req=5 ttl=62 time=0.117 ms
64 bytes from 172.71.36.20: icmp_req=6 ttl=62 time=0.119 ms
^C
--- 172.71.36.20 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4998ms
rtt min/avg/max/mdev = 0.044/0.099/0.167/0.042 ms
root@la1:/tmp/pycore.49574/la1.conf# █

```

Figura 42: Conectividade entre o *laptop* la1 e o *laptop* lb1.

```

root@la1:/tmp/pycore.49574/la1.conf# ping 172.71.38.22
PING 172.71.38.22 (172.71.38.22) 56(84) bytes of data.
64 bytes from 172.71.38.22: icmp_req=1 ttl=62 time=0.060 ms
64 bytes from 172.71.38.22: icmp_req=2 ttl=62 time=0.102 ms
64 bytes from 172.71.38.22: icmp_req=3 ttl=62 time=0.089 ms
64 bytes from 172.71.38.22: icmp_req=4 ttl=62 time=0.114 ms
64 bytes from 172.71.38.22: icmp_req=5 ttl=62 time=0.070 ms
64 bytes from 172.71.38.22: icmp_req=6 ttl=62 time=0.114 ms
^C
--- 172.71.38.22 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4998ms
rtt min/avg/max/mdev = 0.060/0.091/0.114/0.022 ms
root@la1:/tmp/pycore.49574/la1.conf# █

```

Figura 43: Conectividade entre o *laptop* la1 e o *laptop* lc1.

```
root@la1:/tmp/pycore.49574/la1.conf# ping 172.71.40.20
PING 172.71.40.20 (172.71.40.20) 56(84) bytes of data.
64 bytes from 172.71.40.20: icmp_req=1 ttl=61 time=0.086 ms
64 bytes from 172.71.40.20: icmp_req=2 ttl=61 time=0.055 ms
64 bytes from 172.71.40.20: icmp_req=3 ttl=61 time=0.139 ms
64 bytes from 172.71.40.20: icmp_req=4 ttl=61 time=0.142 ms
64 bytes from 172.71.40.20: icmp_req=5 ttl=61 time=0.142 ms
64 bytes from 172.71.40.20: icmp_req=6 ttl=61 time=0.227 ms
^C
--- 172.71.40.20 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4997ms
rtt min/avg/max/mdev = 0.055/0.131/0.227/0.055 ms
root@la1:/tmp/pycore.49574/la1.conf#
```

Figura 44: Conectividade entre o *laptop* la1 e o *laptop* ld1.

Conclusão

A concretização deste trabalho prático permitiu-nos, numa primeira fase, a familiarização de ferramentas como o *Core* e o *Wireshark*, passando pela captura de tráfego IP, análise de datagramas, conhecimento de protocolos, noção de fragmentação ou estudo do cabeçalho IP.

Consideramos o maior obstáculo na primeira parte do ensaio, por um lado, a utilização do *Core* e do *Wireshark*, dada a novidade de trabalho com ferramentas da área de conectividade e endereçamento. Para além disso, achamos assinalar a escassez de tempo, devido à questão de nos acostarmos ao conceito e à interpretação inicial das perguntas, especialmente no que toca à utilização do *software* sugerido.

Já na parte II do trabalho analisamos rotas e tipos de encaminhamento, conceitos como máscara de rede, testes de conectividade, tabelas de encaminhamento ou atribuição dinâmica de endereços, sendo-nos por fim introduzido o conceito de criação de sub-redes.

Nesta parte, o que nos suscitou mais dificuldades foi o *subnetting*, sendo que demoramos a perceber com exatidão os conceitos e o processos associados, acabando por considerarmos a secção de todo o trabalho com maior complexidade em termos concetuais.