

# Trabalho Prático 2 - Protocolo IPv4 (1ª Parte) Redes de Computadores PL53

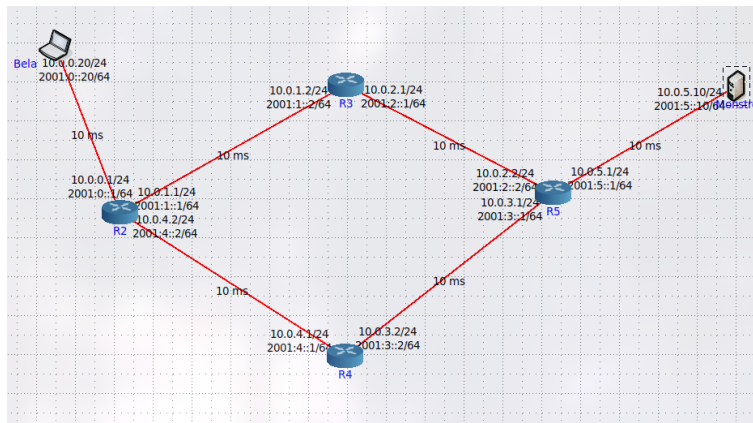
Gonalo Soares<sup>[a93286]</sup>, Mariana Rodrigues<sup>[a93229]</sup>, and Rita Teixeira<sup>[a89494]</sup>

Universidade do Minho

## 1 Parte 1 - Datagramas IP e Fragmentao

### 1.1 Pergunta 1

**Alnea a** O primeiro passo foi construir a topologia CORE. Assim sendo, comeamos por definir o *host (pc)* designado de *Bela* que se encontra conectado ao *router R2*. Por sua vez, este primeiro router foi conectado a dois outros *routers R3, R4* que se encontram conectados ao *router R5*. Por fim, o *host (servidor)* ao qual foi pedido que dssemos o nome de *Monstro* foi conectado a este ltimo router mencionado.



**Fig. 1.** Topologia CORE

Aps a configurao da topologia, ativamos a captura de trfego com o wire-shark no host *Bela*. De seguida, executamos o comando *traceroute -I 10.0.5.10* (sendo o IP 10.0.5.10 o IP do Monstro) na shell do host *Bela* e obtivemos o seguinte resultado.

```

vcmnd
root@Bela:/tmp/pycore.38705/Bela.conf# tracert -I 10.0.5.10
tracert to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 20.222 ms 20.192 ms 20.186 ms
 2 10.0.1.2 (10.0.1.2) 40.513 ms 40.509 ms 40.503 ms
 3 10.0.2.2 (10.0.2.2) 60.712 ms 60.706 ms 60.700 ms
 4 10.0.5.10 (10.0.5.10) 83.936 ms 83.930 ms 83.923 ms
root@Bela:/tmp/pycore.38705/Bela.conf#

```

Fig. 2. Comando *tracert* -I 10.0.5.10 na shell do host *Bela*

**Al nea b** Na seguinte imagem podemos ver tr fego ICMP capturado na shell do host *Bela*:

No.	Time	Source	Destination	Protocol	Length	Info
192	195.796652868	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=3/768, ttl=1 (no response found)
193	195.796650835	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=4/1024, ttl=2 (no response found)
194	195.796668256	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=5/1280, ttl=2 (no response found)
195	195.796714906	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=6/1536, ttl=2 (no response found)
196	195.796682633	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=7/1792, ttl=3 (no response found)
197	195.796690245	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=8/2048, ttl=3 (no response found)
198	195.796697211	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=9/2304, ttl=3 (no response found)
199	195.796735355	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=10/2560, ttl=4 (reply in 225)
200	195.796732454	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=11/2816, ttl=4 (reply in 225)
201	195.796720966	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=12/3072, ttl=4 (reply in 225)
202	195.796720909	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=13/3328, ttl=4 (reply in 227)
203	195.796734509	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=14/3584, ttl=5 (reply in 228)
204	195.796740908	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=15/3840, ttl=5 (reply in 229)
205	195.796740919	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=16/4096, ttl=6 (reply in 230)
206	195.818616200	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
207	195.818613961	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
208	195.818613505	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
209	195.812241847	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=17/4352, ttl=6 (reply in 231)
210	195.812241847	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=18/4608, ttl=6 (reply in 232)
211	195.812250832	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=19/4864, ttl=6 (reply in 233)
212	195.811173001	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
213	195.811172844	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
214	195.811174686	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
215	195.831993908	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=20/5120, ttl=7 (reply in 234)
216	195.832023855	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=21/5376, ttl=7 (reply in 235)
217	195.832022797	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=22/5632, ttl=8 (reply in 236)
218	195.831993908	10.0.0.20	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
219	195.831993944	10.0.0.20	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
220	195.831993944	10.0.0.20	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
221	195.851519162	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=23/6144, ttl=8 (reply in 237)
222	195.851683608	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=24/6400, ttl=8 (reply in 238)
223	195.851686079	10.0.0.20	10.0.5.10	ICMP	74	Echo (ping) request id=0x0024, seq=25/6656, ttl=9 (reply in 239)
224	195.874634833	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=26/6912, ttl=8 (request in 199)
225	195.874638952	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=27/7168, ttl=8 (request in 200)
226	195.874639504	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=28/7424, ttl=8 (request in 201)
227	195.874640141	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=29/7680, ttl=8 (request in 202)
228	195.874640762	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=30/7936, ttl=8 (request in 203)
229	195.874641334	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=31/8192, ttl=8 (request in 204)
230	195.874641815	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=32/8448, ttl=8 (request in 205)
231	195.894874629	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=33/8704, ttl=8 (request in 206)
232	195.894880939	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=34/8960, ttl=8 (request in 207)
233	195.894882512	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=35/9216, ttl=8 (request in 208)
234	195.815230546	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=36/9472, ttl=8 (request in 209)
235	195.815230925	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=37/9728, ttl=8 (request in 210)
236	195.815241448	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=38/9984, ttl=8 (request in 211)
237	195.832627954	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=39/10240, ttl=8 (request in 212)
238	195.832627954	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=40/10496, ttl=8 (request in 213)
239	195.832627954	10.0.0.20	10.0.0.20	ICMP	74	Echo (ping) request id=0x0024, seq=41/10752, ttl=8 (request in 214)
240	196.863887987	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet

Fig. 3. Tr fego capturado depois da execu o do comando *tracert*

Podemos observar que, a partir do primeiro router que encaminharia o datagrama, este devolve uma mensagem de erro para o router origem avisando que o TTL (time-to-live) foi excedido. Os datagramas de erro s o apresentados a preto na imagem. O comando *tracert* manda 3 datagramas com o TTL sempre a crescer comeando a 1 e acabando depois de chegar ao destino. Desta forma   poss vel "descobrir" o caminho que o datagrama percorreu.

Por isso, podemos concluir que os resultados foram os esperados.

**Alínea c** O valor inicial mínimo do campo TTL para alcançar o servidor *Monstro* é 4. De seguida, segue-se a verificação do valores do TTL. Se for um valor de 1 a 3, o pacote será descartado pelo primeiro, segundo e terceiro *routers* e prontamente será enviada a mensagem ICMP, informando a origem que o *time-to-live* foi excedido em trânsito.

Com alguma redundância pois apenas confirmam os resultados obtidos na alínea anterior e apoiados pela topologia criada, executamos para efeitos de verificação o comando *ping*. Assim, este comando foi executado com origem em *Bela* e com destino em *Monstro* e valores de TTL variados, sendo que de seguida serão analisados os resultados.

O comando utilizado é então:

*ping -tx -c110.0.5.10*

A *flag -c* indica o número de pacotes a serem enviados ao endereço IP de destino, que neste caso é 1. A *flag -t* indica o valor do campo TTL do pacote.

Com tudo isto, os resultados de execução são:

```
root@Bela:/tmp/pycore.38705/Bela.conf# ping -t 1 -c 1 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Time to live exceeded

--- 10.0.5.10 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@Bela:/tmp/pycore.38705/Bela.conf# ping -t 2 -c 1 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
From 10.0.1.2 icmp_seq=1 Time to live exceeded

--- 10.0.5.10 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@Bela:/tmp/pycore.38705/Bela.conf# ping -t 3 -c 1 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
From 10.0.2.2 icmp_seq=1 Time to live exceeded

--- 10.0.5.10 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@Bela:/tmp/pycore.38705/Bela.conf#
```

**Fig. 4.** Envio de datagramas com TTL a 1, 2 e a 3

E, por fim, conseguimos alcançar o servidor *Monstro* com o TTL a 4

```

root@Bela:/tmp/pycore.38705/Bela.conf# ping -t 4 -c 1 10.0.5.10 -q
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.

--- 10.0.5.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 80.920/80.920/80.920/0.000 ms
root@Bela:/tmp/pycore.38705/Bela.conf# █

```

**Fig. 5.** Envio do datagrama com TTL a 4

**Alínea d** De maneira a obter o valor médio do tempo de ida-e-volta (RTT - *Round-Trip Time*), é necessário apenas verificar o *output* do comando *traceroute* para o router *Monstro*, que irá mostrar os três tempos obtidos para cada salto. Sendo assim, referente ao tempo de ida-e-volta, temos:

$$RTT = \frac{83.936 + 83.930 + 83.923}{3} = 83.929(6)$$

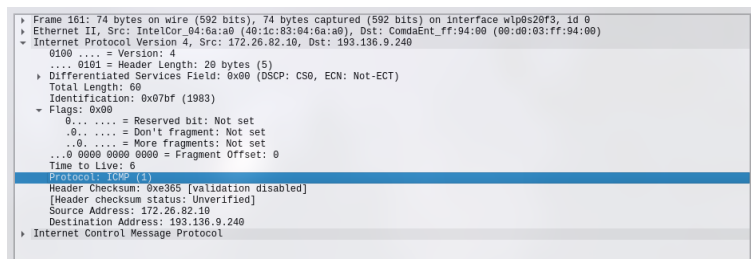
**Alínea e** O *One-way Delay* não irá levar a um cálculo com precisão, pois este método apresenta várias limitações, como, por exemplo, o requisito de co-operação intensiva entre ambos os computadores e também a precisão da medição do atraso que está sujeito a sincronização de precisão.

## 1.2 Pergunta 2

Comecemos por executar:

```
tracert -I router-di.uminho.pt
```

**Alínea a** De maneira a descobrir o endereço IP da interface ativa do nosso computador, devemos localizar o primeiro pacote ICMP enviado, com o intuito de ver qual o IP fonte: 172.26.82.10. Uma outra indicação deste podia ser o facto que é o único IP origem que recebe pacotes TTL *exceeded* (, ou seja, foi a partir deste interface que foi executado o *tracert*).



**Fig. 6.** Informação relativa ao primeiro pacote

**Alínea b** O valor do campo protocolo é ICMP: *Internet Control Message Protocol*. Este é utilizado para controlo a nível de rede e geralmente não possui *payload* útil, isto é, o protocolo ICMP não é utilizado para trocar dados entre sistemas.

**Alínea c** Carregado no primeiro pacote ICMP capturado pelo Wireshark, o tamanho do cabeçalho IPv4 é de 20 bytes. O tamanho do *payload* será o tamanho total do pacote menos o tamanho do cabeçalho: **60 - 20 = 40 bytes**.

**Alínea d** De maneira a determinar se o datagrama IP foi fragmentado ou não, devemos proceder ao processo de verificação do valor da *flag more fragments* e do campo *fragment offset*. Assim sendo, o datagrama IP não foi fragmentado visto que o seu tamanho não excedia o valor do MTU (tipicamente 1500 bytes).

**Al nea e** Analisando os pacotes enviados pela interface ativa do computador, os campos do cabealho IP que variam s o:

- o TTL (*time-to-live*), que   incrementado a cada tr s pacotes;
- o ID  nico de cada pacote;
- o *header checksum*

No.	Time	Source	Destination	Protocol	Length	Info
29	3.343299905	172.16.115.252	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
33	3.343299318	172.16.115.252	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
37	3.343299407	172.16.115.252	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
26	3.325427594	172.16.2.1	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
28	3.343296799	172.16.2.1	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
35	3.343297063	172.16.2.1	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
39	3.343297174	172.26.254.254	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
34	3.343297849	172.26.254.254	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
38	3.343298008	172.26.254.254	172.26.82.10	ICMP	78	Time-to-live exceeded (Time to live exceeded)
2	0.000042011	172.26.82.10	193.137.16.65	TCP	54	41916 - 442 [ACK] Seq=1 Ack=255 Win=500 Len=0
4	0.079572493	172.26.82.10	34.120.186.93	TLSv1.2	97	Application Data
6	3.266762315	172.26.82.10	193.137.16.65	DNS	75	Standard query 0x3ca4 A marco.uminho.pt
7	3.266769406	172.26.82.10	193.137.16.65	DNS	75	Standard query 0x6aa1 AAAA marco.uminho.pt
10	3.308536089	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=1/256, tt
11	3.308554843	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=2/512, tt
12	3.308561716	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=3/768, tt
13	3.308568593	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=4/1024, t
14	3.308574492	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=5/1280, t
15	3.308581218	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=6/1536, t
16	3.308587987	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=7/1792, t
17	3.308593717	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=8/2048, t
18	3.308599713	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=9/2304, t
19	3.308606049	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=10/2560, t
20	3.308612314	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=11/2816, t
21	3.308618426	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=12/3072, t
22	3.308624875	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=13/3328, t
23	3.308630900	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=14/3584, t
24	3.308636603	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=15/3840, t
25	3.308642110	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=16/4096, t
27	3.325524112	172.26.82.10	193.136.9.240	ICMP	526	Echo (ping) request id=0x0008, seq=17/4352, t
44	3.343934811	172.26.82.10	193.137.16.65	DNS	83	Standard query 0x2a93 PTR 1.2.16.172.in-addr.
46	3.374680063	172.26.82.10	193.137.16.145	DNS	83	Standard query 0x2a93 PTR 1.2.16.172.in-addr.
48	3.393171676	172.26.82.10	193.137.16.75	DNS	83	Standard query 0x2a93 PTR 1.2.16.172.in-addr.
50	3.411492292	172.26.82.10	193.137.16.65	DNS	83	Standard query 0x2a93 PTR 1.2.16.172.in-addr.
52	3.437812866	172.26.82.10	193.137.16.145	DNS	83	Standard query 0x2a93 PTR 1.2.16.172.in-addr.
54	3.468749638	172.26.82.10	193.137.16.75	DNS	83	Standard query 0x2a93 PTR 1.2.16.172.in-addr.
56	3.489828095	172.26.82.10	193.137.16.65	DNS	87	Standard query 0x908d PTR 252.115.16.172.in-a
58	3.510193843	172.26.82.10	193.137.16.145	DNS	87	Standard query 0x908d PTR 252.115.16.172.in-a
60	3.515175613	172.26.82.10	193.137.16.75	DNS	87	Standard query 0x908d PTR 252.115.16.172.in-a
62	3.532873177	172.26.82.10	193.137.16.65	DNS	87	Standard query 0x908d PTR 252.115.16.172.in-a
64	3.554780227	172.26.82.10	193.137.16.145	DNS	87	Standard query 0x908d PTR 252.115.16.172.in-a

Fig. 7. Pacotes ICMP capturados

**Al nea f** Atrav s da observa o da informa o do header IP dos v rios pacotes que enviamos, podemos concluir que o campo de identifica o do datagrama   usado para identificar unicamente cada grupo de fragmentos que um dado datagrama tenha sido obrigado a ser descomposto. Assim   poss vel juntar os fragmentos de um datagrama quando estes tiverem chegado ao seu destino.

Quanto ao TTL (time to leave) podemos verificar que por *default* o comando *traceroute* cria desde 3 pacotes com o TTL a 1, a 3 pacotes com o TTL necess rio para o pacote chegar ao destino, incrementando em 1 unidade o TTL. De notar que s o mandados 3 pacotes com o mesmo TTL, de modo, a prever tempos anormais que a rede possa ter que geralmente n o revelaram uma boa estimativa.

Nas imagens a seguir, podemos observar os diferentes valores do campo de identifica o e TTL e como estes se conjugam.

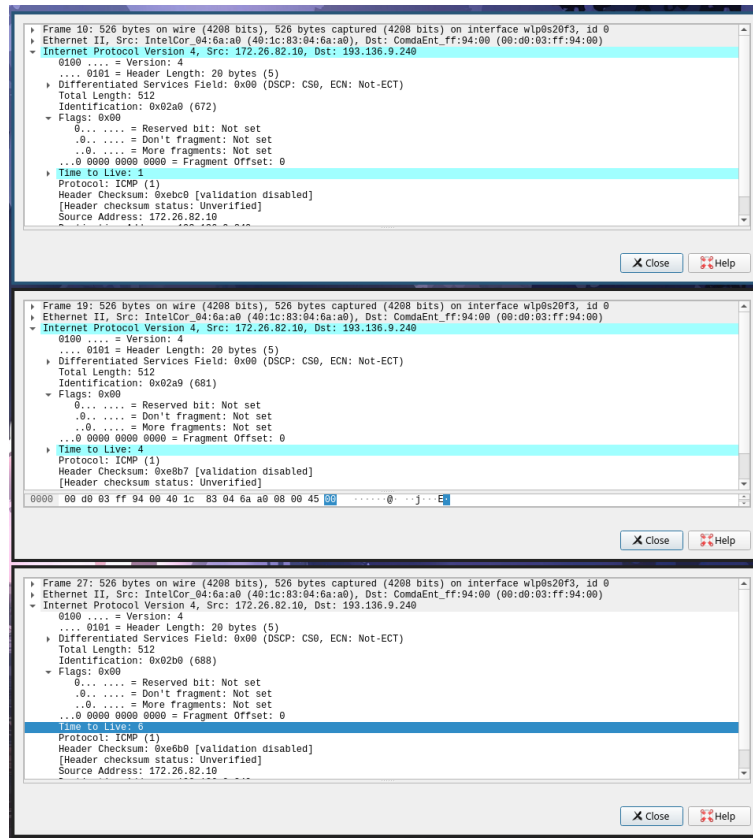
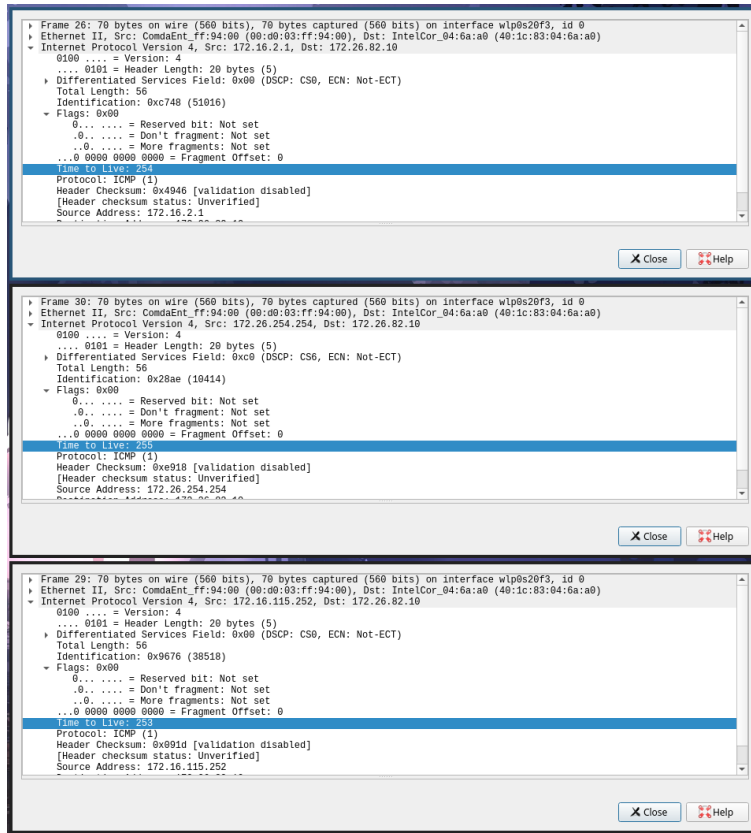


Fig. 8. Informação relativa aos pacotes

**Al nea g** O campo TTL, dos pacotes relativos   s rie de respostas ICMP TTL *exceeded*   igual entre si e toma o valor de 64. Este valor permanece constante para todas as mensagens de resposta ICMP TTL *exceeded* enviadas ao nosso *host* pois   um valor por defeito que provem e   definido pelo pr prio *router* de destino.

No.	Time	Source	Destination	Protocol	Length	Info
26	3.325427504	172.26.82.10	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
28	3.343296790	172.16.2.1	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
29	3.343296965	172.16.115.252	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
30	3.343297074	172.26.254.254	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
33	3.343299318	172.16.115.252	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
34	3.343297949	172.26.254.254	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
35	3.343297063	172.16.2.1	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
37	3.343299497	172.16.115.252	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)
38	3.343298008	172.26.254.254	172.26.82.10	ICMP	70	Time-to-live exceeded (Time to live exceeded)

**Fig. 9.** Pacotes do tipo ICMP TTL *exceeded* capturados



**Fig. 10.** Informa  es relativas aos pacotes capturados



### 1.3 Pergunta 3

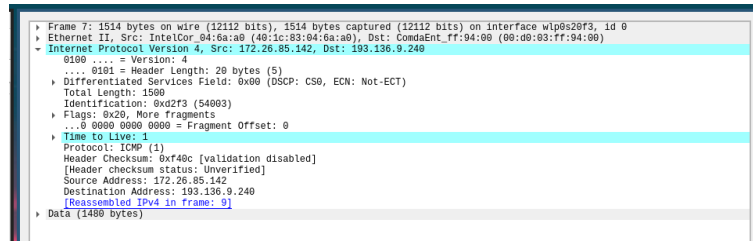
**Alínea a** Após uma pequena observação podemos verificar que a primeira mensagem ICMP se encontra identificada na imagem a seguir com o número 9.

No.	Time	Source	Destination	Protocol	Length	Info
1.8.800000	172.26.85.142	192.168.1.1	192.168.1.1	TCP	54	50808 -> 443 [ACK] Seq=1 Ack=356 Win=496 Len=0
2.1.128426	172.26.85.142	192.168.1.1	192.168.1.1	DNS	75	Standard query 817cbe A marco.uninho.pt
3.1.128444	172.26.85.142	192.168.1.1	192.168.1.1	DNS	75	Standard query 817cbe A marco.uninho.pt
5.2.132184	192.168.1.1	172.26.85.142	172.26.85.142	DNS	129	Standard query response 817cbe A marco.uninho.pt A 192.168.1.248 NS dns2.uninho.pt NS dns1.uninho.pt NS ns82.fcnn.pt NS dns3.uninho.pt...
6.2.132184	192.168.1.1	172.26.85.142	172.26.85.142	DNS	129	Standard query response 817cbe A marco.uninho.pt A 192.168.1.248 NS dns2.uninho.pt NS dns1.uninho.pt NS ns82.fcnn.pt NS dns3.uninho.pt...
7.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 40)
8.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 40)
9.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
10.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
11.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
12.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
13.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
14.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
15.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
16.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
17.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
18.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
19.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
20.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
21.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
22.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
23.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
24.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
25.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
26.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
27.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
28.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
29.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
30.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
31.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
32.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
33.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
34.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
35.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
36.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
37.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
38.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
39.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
40.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
41.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
42.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
43.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
44.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
45.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
46.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
47.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
48.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)
49.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Fragmented IP protocol (protocol=1, offset=0, len=15) (reassembled in 41)
50.2.132178	172.26.85.142	192.168.1.1	192.168.1.1	ICMP	15	Echo (ping) request (id=8000, seq=7200, ttl=1) (no response found)

**Fig. 11.** Captura de Tráfego gerado pelo comando traceroute com pacotes de tamanho 4053 bytes.

Foi necessário fragmentar o pacote inicial visto que ele tinha um tamanho que excedia o MTU. O MTU ou *Maximum transmission unit* é o máximo tamanho que um datagrama poderá ter, de modo a ser enviado pela rede. Tipicamente, o valor do MTU em redes *wifi* ou *ethernet* é de 1500 betys. Visto que o tamanho dos pacotes que enviamos eram de 4053 bytes, podemos concluir que este precisava de ser fragmentado 3 vezes. De notar que ao calcular o número de fragmentos não nós nos podíamos esquecer que o cabeçalho do protocolo IPv4 ocupa 20 bytes.

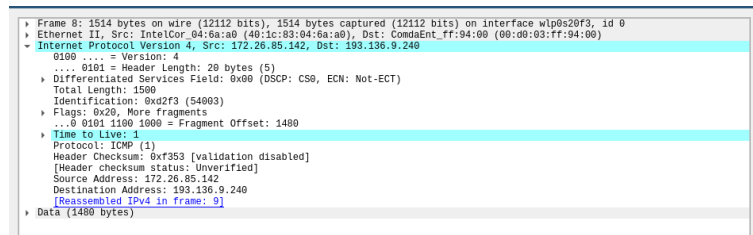
**Al nea b** Na figura seguinte podemos observar as informa es relativas ao primeiro fragmento.



**Fig. 12.** Informa es relativas ao primeiro fragmento.

Atrav s do campo *flags*, podemos ver que a flag *More fragments* est  marcada a verdadeiro. Tamb m, o campo *Fragment Offset* encontra-se a 0. Logo, podemos concluir que para al m do datagrama ter sido fragmentado, este   o primeiro fragmento. O tamanho do datagrama   de 1500 bytes.

**Al nea c** De seguida, podemos ver as as informa es relativas ao segundo fragmento do datagrama IP original.



**Fig. 13.** Informa es relativas ao segundo fragmento.

Podemos verificar que n o se trata do primeiro fragmento visto que o campo *Fragment Offset*   diferente de zero. Tamb m, h  mais fragmentos para al m deste visto que a flag *More fragments* est  assinalada.

**Alínea d** Foram criados 3 fragmentos a partir do datagrama original.

**Alínea e** Os campos que variam entre os diferentes segmentos são a flag *More fragments* que indica se há mais fragmentos a ser recebidos depois deste e o *Fragment Offset* que permite saber o número de bytes que precedem este fragmento. Quando todos os fragmentos chegam ao destino, é através do *Fragment Offset* que é deduzida a ordem destes. Neste caso, o pacote com offset 0 é o primeiro. O pacote com offset 1480 é o segundo e depois temos o ultimo pacote que podemos identificar tanto pela flag *More fragments* (que está a 0) ou por ser o fragmento com maior *Fragment Offset*.

**Alínea f** Podemos verificar o processo de fragmentação através do seguinte calculo:

Primeiro calculamos o número de fragmentos com o MTU (1500) preenchido: Não esquecer de retirar o tamanho do header do IP ao efetuar os cálculos.

$$4053 / MTU = 2$$

Depois avaliamos se existe ainda um pacote que tenha de ser preenchido com um valor menor que o MTU:

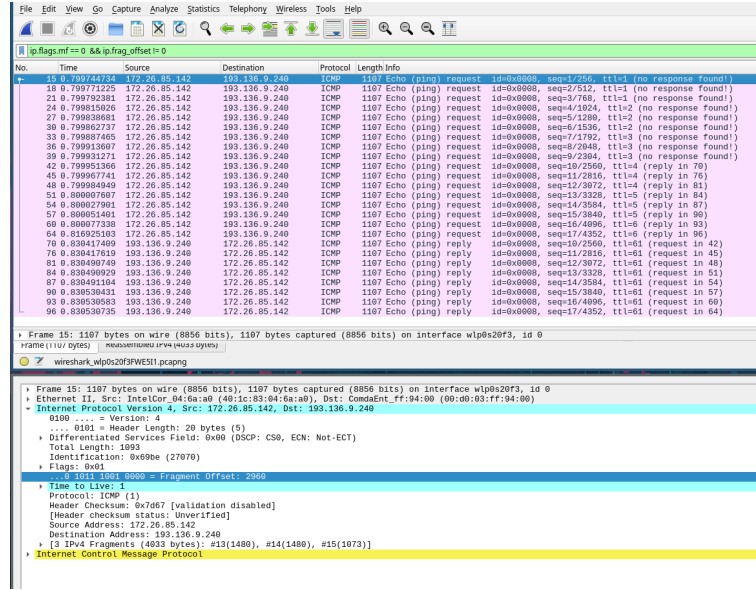
$$4053 \% MTU = 1093$$

Temos que voltar a considerar o tamanho do header do IP. Logo temos os dois fragmentos iniciais com tamanho de 1500 bytes e o ultimo com tamanho 1113 bytes.

**Al nea g** Podemos detetar o  ltimo fragmento com um dado *ID* atrav s da seguinte express o l gica:

$$moreFragments == 0 \ \&\& \ fragmentOffset != 0$$

Na imagem seguinte convertemos a express o acima em um filtro para mostrar s  os  ltimos fragmentos no wireshark:

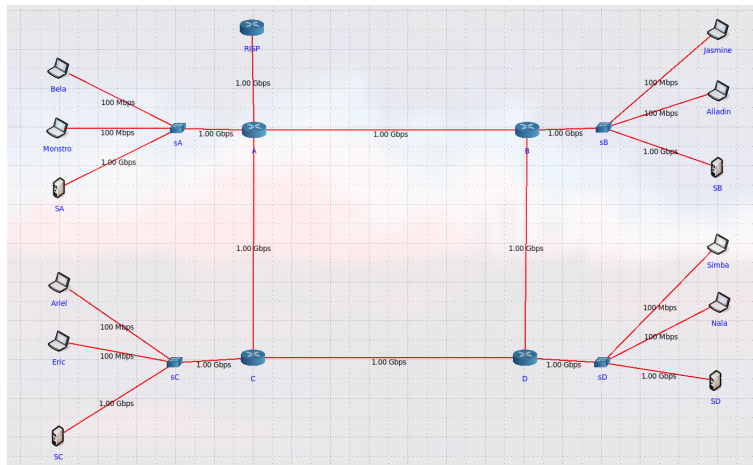


**Fig. 14.** Visualiza  o dos  ltimos fragmentos no *wireshark*

## 2 Parte 2 - Endereçamento e Encaminhamento

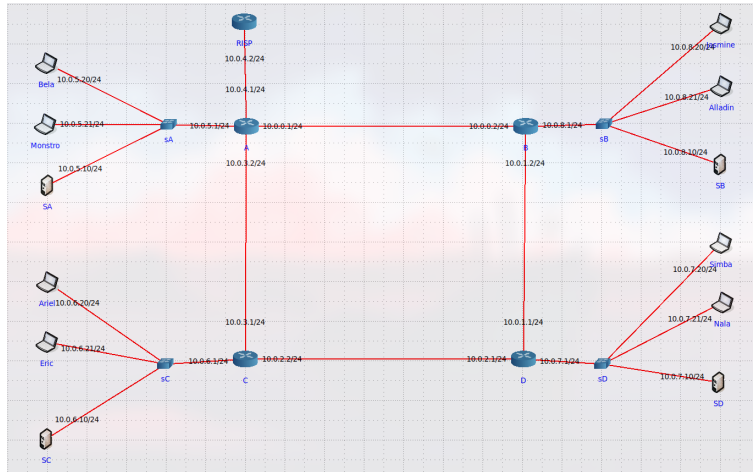
### 2.1 Pergunta 1

Iniciamos o processo da segunda parte do relatório, com a construção da seguinte topologia:



**Fig. 15.** Topologia utilizada e nomes de cada dispositivo

**Alínea a** Os endereços IP e as máscaras de rede atribuídos, podem ser observados na seguinte imagem:



**Fig. 16.** Visualizao dos endereos IP e mscaras de rede da topologia

A mscara de rede ser 111111.11111111.11111111.00000000 ou 55.55.255.0 , o que corresponde a /24 em notaco CIDR.

**Alnea b** De acordo com a norma **RFC** 1918, as gamas de endereos **IP** privados so os seguintes:

- 192.168.0.0 - 192.168.255.255
- 172.16.0.0 - 172.16.255.255
- 10.0.0.0 - 10.255.255.255

Pelo que, pode ser concluido que os endereos obtidos na topologia **core** so endereos **privados**.

**Alnea c** No  atribuido um endereo IP aos *switches* visto que estes trabalham sobre a camada 2 (Data Link Layer).

**Alínea d** De maneira a verificar a conectividade **IP** interna a cada departamento, entre cada laptop e o servidor respectivo. Seguem-se a visualização dos resultados do comando **ping** realizado:

```

vcmd
root@Bela:/tmp/pycore.33097/Bela.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=64 time=0.931 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=0.176 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=64 time=0.190 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=64 time=0.157 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=64 time=0.151 ms
64 bytes from 10.0.5.10: icmp_seq=6 ttl=64 time=0.194 ms
64 bytes from 10.0.5.10: icmp_seq=7 ttl=64 time=0.167 ms
^C
--- 10.0.5.10 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6119ms
rtt min/avg/max/ndev = 0.151/0.280/0.351/0.265 ms
root@Bela:/tmp/pycore.33097/Bela.conf#

vcmd
root@Jasmine:/tmp/pycore.33097/Jasmine.conf# ping 10.0.8.10
PING 10.0.8.10 (10.0.8.10) 56(84) bytes of data:
64 bytes from 10.0.8.10: icmp_seq=1 ttl=64 time=0.919 ms
64 bytes from 10.0.8.10: icmp_seq=2 ttl=64 time=0.189 ms
64 bytes from 10.0.8.10: icmp_seq=3 ttl=64 time=0.189 ms
64 bytes from 10.0.8.10: icmp_seq=4 ttl=64 time=0.179 ms
64 bytes from 10.0.8.10: icmp_seq=5 ttl=64 time=0.204 ms
64 bytes from 10.0.8.10: icmp_seq=6 ttl=64 time=0.154 ms
64 bytes from 10.0.8.10: icmp_seq=7 ttl=64 time=0.305 ms
^C
--- 10.0.8.10 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6111ms
rtt min/avg/max/ndev = 0.154/0.697/3.050/0.393 ms
root@Jasmine:/tmp/pycore.33097/Jasmine.conf#

vcmd
root@Eric:/tmp/pycore.33097/Eric.conf# ping 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data:
64 bytes from 10.0.6.10: icmp_seq=1 ttl=64 time=1.40 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=64 time=0.195 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=64 time=0.181 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=64 time=0.162 ms
64 bytes from 10.0.6.10: icmp_seq=5 ttl=64 time=0.147 ms
64 bytes from 10.0.6.10: icmp_seq=6 ttl=64 time=0.167 ms
64 bytes from 10.0.6.10: icmp_seq=7 ttl=64 time=0.147 ms
64 bytes from 10.0.6.10: icmp_seq=8 ttl=64 time=0.143 ms
64 bytes from 10.0.6.10: icmp_seq=9 ttl=64 time=0.145 ms
^C
--- 10.0.6.10 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8166ms
rtt min/avg/max/ndev = 0.143/0.238/1.397/0.368 ms
root@Eric:/tmp/pycore.33097/Eric.conf#

vcmd
root@Nala:/tmp/pycore.33097/Nala.conf# ping 10.0.7.10
PING 10.0.7.10 (10.0.7.10) 56(84) bytes of data:
64 bytes from 10.0.7.10: icmp_seq=1 ttl=64 time=0.632 ms
64 bytes from 10.0.7.10: icmp_seq=2 ttl=64 time=0.172 ms
64 bytes from 10.0.7.10: icmp_seq=3 ttl=64 time=0.098 ms
64 bytes from 10.0.7.10: icmp_seq=4 ttl=64 time=0.143 ms
64 bytes from 10.0.7.10: icmp_seq=5 ttl=64 time=4.95 ms
64 bytes from 10.0.7.10: icmp_seq=6 ttl=64 time=0.226 ms
64 bytes from 10.0.7.10: icmp_seq=7 ttl=64 time=0.151 ms
64 bytes from 10.0.7.10: icmp_seq=8 ttl=64 time=0.129 ms
64 bytes from 10.0.7.10: icmp_seq=9 ttl=64 time=0.153 ms
^C
--- 10.0.7.10 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8139ms
rtt min/avg/max/ndev = 0.098/0.695/4.554/1.372 ms
root@Nala:/tmp/pycore.33097/Nala.conf#

```

**Fig. 17.** Visualização da conectividade **IP** interna de cada departamento

**Alínea e** De maneira a verificar a conectividade **IP** externa entre os departamentos, seguem-se a visualização dos resultados do comando **ping** realizado:

```

root@Bela:/tmp/pycore.42793/Bela.conf# ping 10.0.8.20
PING 10.0.8.20 (10.0.8.20) 56(84) bytes of data.
64 bytes from 10.0.8.20: icmp_seq=1 ttl=62 time=1.71 ms
64 bytes from 10.0.8.20: icmp_seq=2 ttl=62 time=0.301 ms
64 bytes from 10.0.8.20: icmp_seq=3 ttl=62 time=0.295 ms
^C
--- 10.0.8.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2024ms
rtt min/avg/max/mdev = 0.295/0.768/1.708/0.664 ms
root@Bela:/tmp/pycore.42793/Bela.conf# ping 10.0.7.20
PING 10.0.7.20 (10.0.7.20) 56(84) bytes of data.
64 bytes from 10.0.7.20: icmp_seq=1 ttl=61 time=2.27 ms
64 bytes from 10.0.7.20: icmp_seq=2 ttl=61 time=0.373 ms
64 bytes from 10.0.7.20: icmp_seq=3 ttl=61 time=0.400 ms
64 bytes from 10.0.7.20: icmp_seq=4 ttl=61 time=0.349 ms
^C
--- 10.0.7.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3048ms
rtt min/avg/max/mdev = 0.349/0.848/2.271/0.821 ms
root@Bela:/tmp/pycore.42793/Bela.conf# ping 10.0.6.20
PING 10.0.6.20 (10.0.6.20) 56(84) bytes of data.
64 bytes from 10.0.6.20: icmp_seq=1 ttl=62 time=1.84 ms
64 bytes from 10.0.6.20: icmp_seq=2 ttl=62 time=0.207 ms
64 bytes from 10.0.6.20: icmp_seq=3 ttl=62 time=0.297 ms
^C
--- 10.0.6.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 0.207/0.782/1.844/0.751 ms
root@Bela:/tmp/pycore.42793/Bela.conf# █

```

**Fig. 18.** Visualizao da conectividade externa entre os departamentos

**Alnea f** Segue-se a verificao da conectividade **IP** entre o porttil *Bela* e o router de acesso *RISP*:



```

root@Bela:/tmp/pycore.33097/Bela.conf# ping 10.0.4.2
PING 10.0.4.2 (10.0.4.2) 56(84) bytes of data:
64 bytes from 10.0.4.2: icmp_seq=1 ttl=63 time=0.996 ms
64 bytes from 10.0.4.2: icmp_seq=2 ttl=63 time=0.207 ms
64 bytes from 10.0.4.2: icmp_seq=3 ttl=63 time=0.214 ms
64 bytes from 10.0.4.2: icmp_seq=4 ttl=63 time=0.251 ms
64 bytes from 10.0.4.2: icmp_seq=5 ttl=63 time=1.12 ms
64 bytes from 10.0.4.2: icmp_seq=6 ttl=63 time=0.243 ms
^C
--- 10.0.4.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5080ms
rtt min/avg/max/mdev = 0.207/0.505/1.121/0.393 ms
root@Bela:/tmp/pycore.33097/Bela.conf#

```

Fig. 19. Visualização da conectividade IP entre o portátil *Bela* e o router de acesso *RISP*

## 2.2 Pergunta 2

Alínea a Execução do comando **netstat -rn** para o *router* e o portátil *bela*, respetivamente:

```

root@RISP:/tmp/pycore.33097/R.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 0.0.0.0 255.255.255.0 U 0 0 0 eth3
10.0.6.0 10.0.3.1 255.255.255.0 UG 0 0 0 eth2
10.0.7.0 10.0.0.2 255.255.255.0 UG 0 0 0 eth0
10.0.8.0 10.0.0.2 255.255.255.0 UG 0 0 0 eth0
root@RISP:/tmp/pycore.33097/R.conf#

root@Bela:/tmp/pycore.33097/Bela.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@Bela:/tmp/pycore.33097/Bela.conf#

```

Fig. 20. Comando **netstat -rn**

Como podemos observar, as **tabelas de routing** ajudam cada um dos dispositivos a descobrirem qual será o próximo salto a ser tomado que um respetivo pacote deverá percorrer até chegar ao seu destino.

Na coluna:

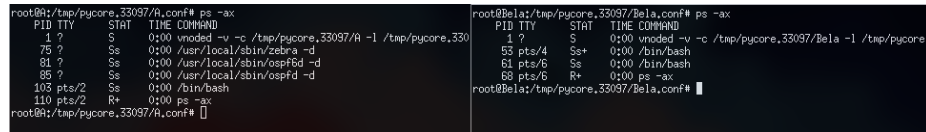
- **Destination** é nos indicada a sub-rede destino
- **Gateway** informa-nos por qual dispositivo o pacote irá passar.
- **Genmask** indica o tipo de máscara utilizada.

No caso do portátil **Bela**, existem apenas duas possibilidades. A primeira é independente do endereço de destino, na qual o pacote irá para o router *A*. Já a segunda, indica que o pacote destino encontra-se dentro da sub-rede do Departamento, pelo que pode optar por um destino.

No caso do router **A**, quando o *Gateway* encontra-se com o valor 0.0.0.0, o pacote pode seguir um qualquer caminho. Os pacotes que tenham como destino um dispositivo da sub-rede 10.0.1.0 , 10.0.7.0 e 10.0.8.0 tero que passar pelo router **B** (10.0.0.2). J quanto aos pacotes com destino nas sub-redes 10.0.2.0 e 10.0.6.0 iro passar pelo router **C** (10.0.3.1).

**Alnea b** Como pode ser observado na imagem abaixo, podemos constatar que no *router RA* o encaminhamento  dinmico visto que na coluna *COMMAND* pode-se verificar que  utilizado o protocolo **ospfd**. Sendo com isto possvel, que um qualquer pacote siga diferentes caminhos quando no  possvel seguir a rota esperada.

J no porttil *Bela* o encaminhamento  esttico dado que no  usado nenhum protocolo.



```

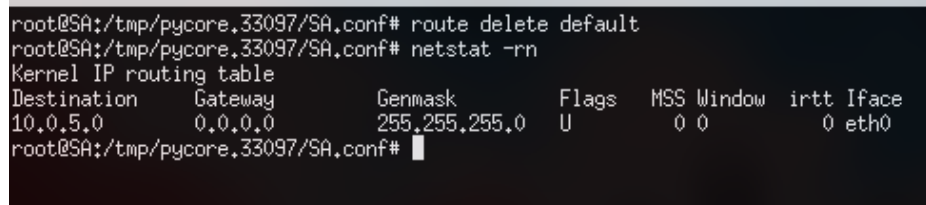
root@RA:/tmp/pycore,33097/RA.conf# ps -ax
  PID TTY          STAT TIME COMMAND
    1 ?        S      0:00 init [v] -c /tmp/pycore,33097/RA -l /tmp/pycore,330
   75 ?        Ss     0:00 /usr/local/sbin/zebra -d
   81 ?        Ss     0:00 /usr/local/sbin/ospfd -d
   88 ?        Ss     0:00 /usr/local/sbin/ospfd -d
  105 pts/2    Ss     0:00 /bin/bash
  110 pts/2    R+     0:00 ps -ax
root@RA:/tmp/pycore,33097/RA.conf#

root@Bela:/tmp/pycore,33097/Bela.conf# ps -ax
  PID TTY          STAT TIME COMMAND
    1 ?        S      0:00 init [v] -c /tmp/pycore,33097/Bela -l /tmp/pycore,
   53 pts/4    Ss+    0:00 /bin/bash
   61 pts/6    Ss     0:00 /bin/bash
   68 pts/6    R+     0:00 ps -ax
root@Bela:/tmp/pycore,33097/Bela.conf#

```

**Fig. 21.** Comando *ps -ax*

**Alnea c** Na shell do servidor **SA**, comeamos por eliminar a rota por defeito da tabela de encaminhamento:



```

root@SA:/tmp/pycore,33097/SA.conf# route delete default
root@SA:/tmp/pycore,33097/SA.conf# netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags   MSS Window  irtt Iface
10.0.5.0            0.0.0.0            255.255.255.0     U        0 0          0 eth0
root@SA:/tmp/pycore,33097/SA.conf#

```

**Fig. 22.** Eliminao da rota por defeito da tabela de encaminhamento do servidor *SA*

Ao ser retirada esta rota da tabela de encaminhamento, o servidor passa a s conseguir comunicar com dispositivos que se encontrem na sua rede.

Pode ser possvel confirmar isso, atravs da seguinte figura:

```
root@SA:/tmp/pycore.33097/SA.conf# ping 10.0.0.2
ping: connect: Network is unreachable
root@SA:/tmp/pycore.33097/SA.conf#
```

**Fig. 23.** Teste de conectividade de rede a um dispositivo que não se encontra na sua rede

**Alínea d** De modo a ser possível restaurar as ligações entre os restantes dispositivos foram executados os seguintes comandos:

```
root@SA:/tmp/pycore.33097/SA.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.5.1
root@SA:/tmp/pycore.33097/SA.conf# route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.5.1
root@SA:/tmp/pycore.33097/SA.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.5.1
root@SA:/tmp/pycore.33097/SA.conf# route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.5.1
```

**Fig. 24.** Adição das rotas estáticas necessárias

Através da adição das rotas mencionadas anteriormente, o *router* passa a saber reencaminhar os pacotes, reestabelecendo assim as ligações.

**Alínea e** De modo a ser possível comprovar que as ligações encontram-se reestabelecidas foram efetuados diversos **ping's** *request's*, como se pode observar em baixo:

```

root@SA:/tmp/pycore.33097/SA.conf# ping 10.0.7.10
PING 10.0.7.10 (10.0.7.10) 56(84) bytes of data.
64 bytes from 10.0.7.10: icmp_seq=1 ttl=61 time=1.26 ms
64 bytes from 10.0.7.10: icmp_seq=2 ttl=61 time=0.406 ms
^C
--- 10.0.7.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.406/0.832/1.259/0.426 ms
root@SA:/tmp/pycore.33097/SA.conf# ping 10.0.8.20
PING 10.0.8.20 (10.0.8.20) 56(84) bytes of data.
64 bytes from 10.0.8.20: icmp_seq=1 ttl=62 time=1.16 ms
64 bytes from 10.0.8.20: icmp_seq=2 ttl=62 time=0.348 ms
^C
--- 10.0.8.20 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.348/0.751/1.155/0.403 ms
root@SA:/tmp/pycore.33097/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=62 time=1.15 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=62 time=0.319 ms
^C
--- 10.0.6.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.319/0.732/1.145/0.413 ms
root@SA:/tmp/pycore.33097/SA.conf# ping 10.0.4.2
PING 10.0.4.2 (10.0.4.2) 56(84) bytes of data.
64 bytes from 10.0.4.2: icmp_seq=1 ttl=63 time=0.686 ms
64 bytes from 10.0.4.2: icmp_seq=2 ttl=63 time=0.105 ms
^C
--- 10.0.4.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 0.105/0.395/0.686/0.290 ms
root@SA:/tmp/pycore.33097/SA.conf# 

```

**Fig. 25.** Resultados obtidos na execuo do comando *ping*

Segue-se a nova tabela de *routing* referente ao servidor:

```

root@SA:/tmp/pycore.33097/SA.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.4.0          10.0.5.1        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 eth0
10.0.6.0          10.0.5.1        255.255.255.0   UG      0 0        0 eth0
10.0.7.0          10.0.5.1        255.255.255.0   UG      0 0        0 eth0
10.0.8.0          10.0.5.1        255.255.255.0   UG      0 0        0 eth0
root@SA:/tmp/pycore.33097/SA.conf# █

```

Fig. 26. Nova Tabela de encaminhamento do servidor

### 2.3 Pergunta 3

**Alínea a** Para o endereço **IP** 192.168.053.128/25, sobram-nos 7 bits para gerirmos as sub-redes. Sabendo que, tendo  $n$  bits reservados para subnetting teremos  $2^n$  sub-redes. Para ser possível suportar toda a topologia atual, apenas seriam precisos 2 bits.

$$\begin{aligned}
 2^n &\geq 4 &<=> \\
 n &\geq \log_2 4 &<=> \\
 n &\geq 2
 \end{aligned}$$

Visto que o número de departamentos pode vir a aumentar, decidimos reservar 3 bits.

- Departamento A :
  - Sub-Rede: 192.168.53.128
  - Gama de valores: 192.168.53.129 - 192.168.53.142
- Departamento B :
  - Sub-Rede: 192.168.53.144
  - Gama de valores: 192.168.53.145 - 192.168.53.158
- Departamento C :
  - Sub-Rede: 192.168.53.160
  - Gama de valores: 192.168.53.161 - 192.168.53.174
- Departamento D :
  - Sub-Rede: 192.168.53.176
  - Gama de valores: 192.168.53.177 - 192.168.53.190

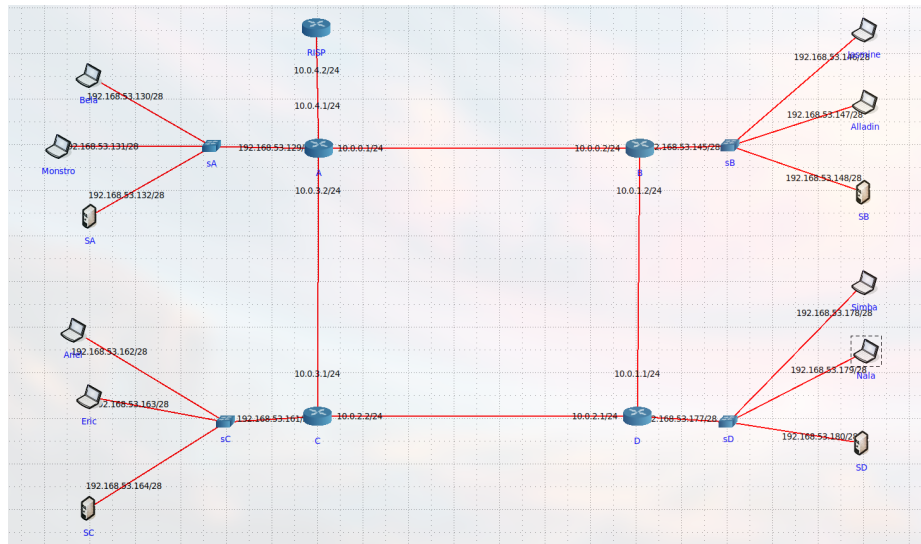


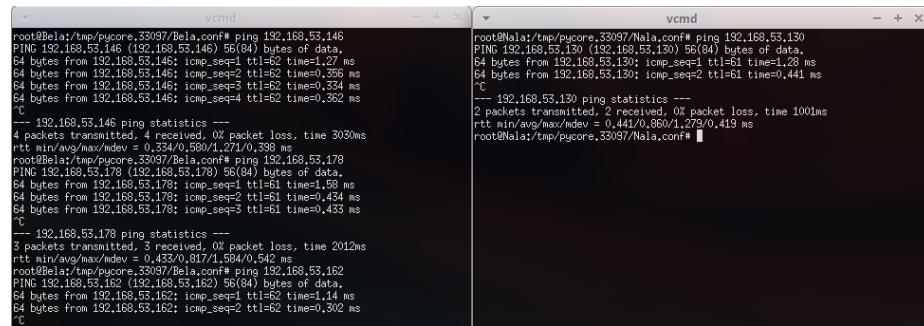
Fig. 27. Topologia do novo endereamento

**Alnea b** A mscara de rede usada foi a de /28, na notao *CIDR*, corresponde a 255.255.255.240.

Sendo que com uma mscara de /28 temos 4 bits disponveis para o *host*, logo a quantidade de *hosts IP* que se podem interligar em cada departamento ser de  $2^4 - 2$ , ou seja, 14 *hosts*.

Visto estarmos a reservar 3 bits para as sub-redes e neste momento s estarem a ser utilizadas 4 sub-redes, podemos constatar que nos sobram 4 ( $2^{3-4}$ ) prefixos de sub-rede disponveis para o futuro.

**Alínea c** De modo a comprovar a conectividade *IP* entre as várias redes locais, foram efetuados diversos comandos *ping*'s, como se pode observar nas imagens abaixo:



```

vcmd
root@Bela:/tmp/pycore.33097/Bela.conf# ping 192.168.53.146
PING 192.168.53.146 (192.168.53.146) 56(84) bytes of data:
64 bytes from 192.168.53.146: icmp_seq=1 ttl=62 time=1.27 ms
64 bytes from 192.168.53.146: icmp_seq=2 ttl=62 time=0.362 ms
64 bytes from 192.168.53.146: icmp_seq=3 ttl=62 time=0.334 ms
64 bytes from 192.168.53.146: icmp_seq=4 ttl=62 time=0.362 ms
^C
--- 192.168.53.146 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 309ms
rtt min/avg/max/mdev = 0.334/0.580/1.271/0.398 ms
root@Bela:/tmp/pycore.33097/Bela.conf# ping 192.168.53.178
PING 192.168.53.178 (192.168.53.178) 56(84) bytes of data:
64 bytes from 192.168.53.178: icmp_seq=1 ttl=61 time=1.58 ms
64 bytes from 192.168.53.178: icmp_seq=2 ttl=61 time=0.434 ms
64 bytes from 192.168.53.178: icmp_seq=3 ttl=61 time=0.433 ms
^C
--- 192.168.53.178 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 201ms
rtt min/avg/max/mdev = 0.433/0.817/1.584/0.542 ms
root@Bela:/tmp/pycore.33097/Bela.conf# ping 192.168.53.162
PING 192.168.53.162 (192.168.53.162) 56(84) bytes of data:
64 bytes from 192.168.53.162: icmp_seq=1 ttl=62 time=1.14 ms
64 bytes from 192.168.53.162: icmp_seq=2 ttl=62 time=0.302 ms
^C

```

```

vcmd
root@Nala:/tmp/pycore.33097/Nala.conf# ping 192.168.53.130
PING 192.168.53.130 (192.168.53.130) 56(84) bytes of data:
64 bytes from 192.168.53.130: icmp_seq=1 ttl=61 time=1.28 ms
64 bytes from 192.168.53.130: icmp_seq=2 ttl=61 time=0.441 ms
^C
--- 192.168.53.130 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.441/0.860/1.273/0.419 ms
root@Nala:/tmp/pycore.33097/Nala.conf#

```

**Fig. 28.** Testes à conectividade *IP* interna da rede local

### 3 Conclusão

Com a realização deste trabalho prático foi nos possíveis consolidar a matéria dada nas aulas teóricas. Nomeadamente, numa primeira fase o funcionamento do protocolo *IP* e as várias técnicas para melhor analisar e perceber datagramas *IP* e o processo de fragmentação. Já numa segunda fase, solidificamos os conhecimentos das diversas técnicas de endereçamento e de encaminhamento.