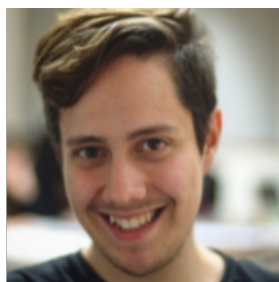




Beatriz Rocha
A84003



Filipe Guimarães
A85308



Gonçalo Ferreira
A84073

Relatório do Trabalho Prático 2 de Redes de Computadores Grupo 1

12 de Novembro de 2019



Conteúdo

I	5
1 Pergunta 1	6
1.1 Alínea a	7
1.2 Alínea b	7
1.3 Alínea c	8
1.4 Alínea d	8
2 Pergunta 2	9
2.1 Alínea a	10
2.2 Alínea b	10
2.3 Alínea c	10
2.4 Alínea d	10
2.5 Alínea e	11
2.6 Alínea f	12
2.7 Alínea g	12
3 Pergunta 3	13
3.1 Alínea a	14
3.2 Alínea b	14
3.3 Alínea c	14
3.4 Alínea d	15
3.5 Alínea e	16
II	17
1 Pergunta 1	18
1.1 Alínea a	18
1.2 Alínea b	19
1.3 Alínea c	19
1.4 Alínea d	19
1.5 Alínea e	21
2 Pergunta 2	23
2.1 Alínea a	23
2.2 Alínea b	24
2.3 Alínea c	24
2.4 Alínea d	25

2.5	Alínea e	26
3	Pergunta 3	28
3.0.1	Alínea 1	28
3.0.2	Alínea 2	29
3.0.3	Alínea 3	29
4	Conclusões	31

Lista de Figuras

1.1	Topologia CORE	6
1.2	Comando traceroute -I para o endereço IP do host h5	7
1.3	Tráfego ICMP enviado e recebido por s1	8
2.1	Traceroute feito a marco.uminho.pt com o pingplotter	9
2.2	Traceroute feito a marco.uminho.pt para tamanho padrão	9
2.3	Tráfego capturado pelo wireshark	10
2.4	Campo Flags	11
2.5	Pacotes ordenados de acordo com o endereço de IP fonte	11
2.6	Pacotes ordenados de acordo com o endereço de IP destino	12
3.1	Tráfego capturado pelo wireshark para um tamanho de pacote de 4201	13
3.2	Traceroute feito a marco.uminho.pt para tamanho padrão	13
3.3	Primeiro fragmento do datagrama IP segmentado	14
3.4	Segundo fragmento do datagrama IP segmentado	15
3.5	Terceiro fragmento do datagrama IP segmentado	15
1.1	Topologia da organização MIEI-RC	18
1.2	Conectividade entre o laptop do departamento A e o servidor	20
1.3	Conectividade entre o laptop do departamento B e o servidor	20
1.4	Conectividade entre o laptop do departamento C e o servidor	21
1.5	Conectividade entre o laptop do departamento D e o servidor	21
1.6	Conectividade entre o router Rext e o servidor S1	22
2.1	Tabela de encaminhamento de Rb	24
2.2	Tabela de encaminhamento de n23	24
2.3	Protocolos usados	24
2.4	Ping de Ra para S1	25
2.5	Ping de Rb para S1	25
2.6	Adicionar novas rotas de S1 para os respectivos departamentos	26
2.7	Ping de n7 para S1	26
2.8	Ping de n16 para S1	27
2.9	Ping de n20 para S1	27
2.10	Ping de n24 para S1	27
3.1	Teste de conexão (ping) entre um laptop do departamento A e um servidor do mesmo departamento	30
3.2	Teste de conexão (ping) entre um laptop do departamento A e B	30

3.3	Teste de conexão (ping) entre um laptop do departamento A e C	30
3.4	Teste de conexão (ping) entre um laptop do departamento A e D	30

Parte I

Capítulo 1

Pergunta 1

Prepare uma topologia no CORE para verificar o comportamento do traceroute. Ligue um host (servidor) s1 a um router r2; o router r2 a um router r3, o router r3 a um router r4, que por sua vez, se liga a um host (pc) h5. (Note que pode não existir conectividade IP imediata entre s1 e h5 até que o routing estabilize). Ajuste o nome dos equipamentos atribuídos por defeito para a topologia do enunciado.

Implementamos a topologia CORE apresentada na figura 1.1.

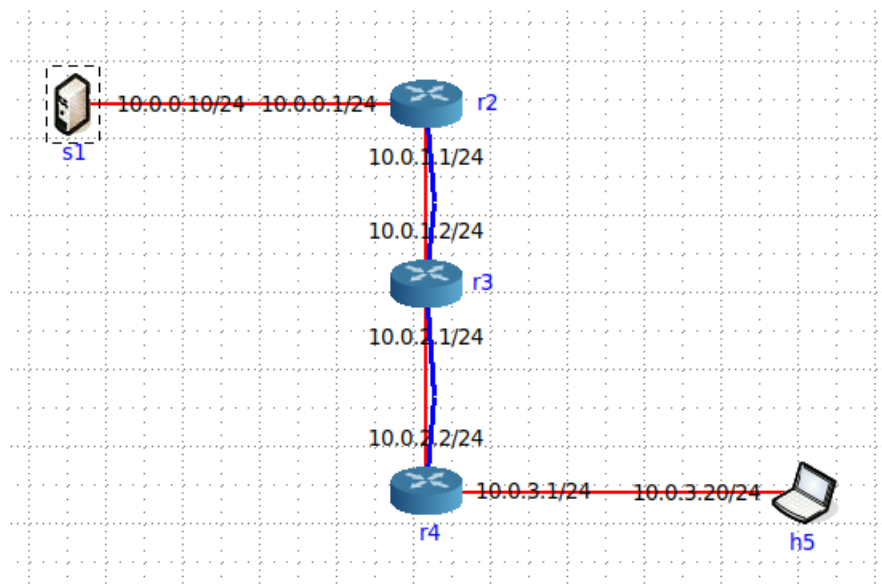
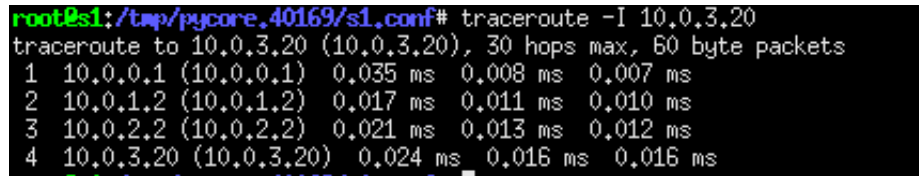


Figura 1.1: Topologia CORE

1.1 Alínea a

Active o wireshark ou o tcpdump no pc s1. Numa shell de s1, execute o comando `tracert -I` para o endereço IP do host h5.

Ativamos o wireshark no pc s1, executando o comando `tracert -I` para o endereço IP do host h5 (10.0.3.20), tal como se pode observar na figura 1.2.



```
root@s1:/tmp/pycore.40169/s1.conf# tracert -I 10.0.3.20
tracert to 10.0.3.20 (10.0.3.20), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.035 ms 0.008 ms 0.007 ms
 2 10.0.1.2 (10.0.1.2) 0.017 ms 0.011 ms 0.010 ms
 3 10.0.2.2 (10.0.2.2) 0.021 ms 0.013 ms 0.012 ms
 4 10.0.3.20 (10.0.3.20) 0.024 ms 0.016 ms 0.016 ms
```

Figura 1.2: Comando `tracert -I` para o endereço IP do host h5

1.2 Alínea b

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

Esperávamos que enquanto o TTL de cada pacote fosse inferior a 4, o pacote não chegasse a h5. Quando o TTL atinge o valor 0, o router descarta o datagrama e devolve uma mensagem de controlo ICMP (Internet Control Message Protocol) ao host de origem, indicando que o TTL foi excedido. Cada vez que passa no router é decrementado o TTL de cada datagrama. Assim, e tendo em conta que existem três routers nesta topologia CORE, seria necessário um TTL mínimo igual a 4. Inicialmente, foram lançados três pacotes com TTL=1 que deveriam chegar a h5, mas que não passaram do router r2, tendo sido emitida uma mensagem de erro. De seguida, foram lançados mais três pacotes com TTL=2 que deveriam chegar a h5, mas que não passaram do router r3, tendo sido emitida uma mensagem de erro. Depois, foram lançados mais três pacotes com TTL=3 que deveriam chegar a h5, mas que não passaram do router r4, tendo sido emitida uma mensagem de erro. O TTL foi incrementado e os pacotes conseguiram chegar ao destino, tal como podemos observar na figura 1.3. Deste modo, verificamos que é recebida uma mensagem de erro ("Time-to-live exceeded") sempre que o TTL é inferior a 4, tal como prevíamos.

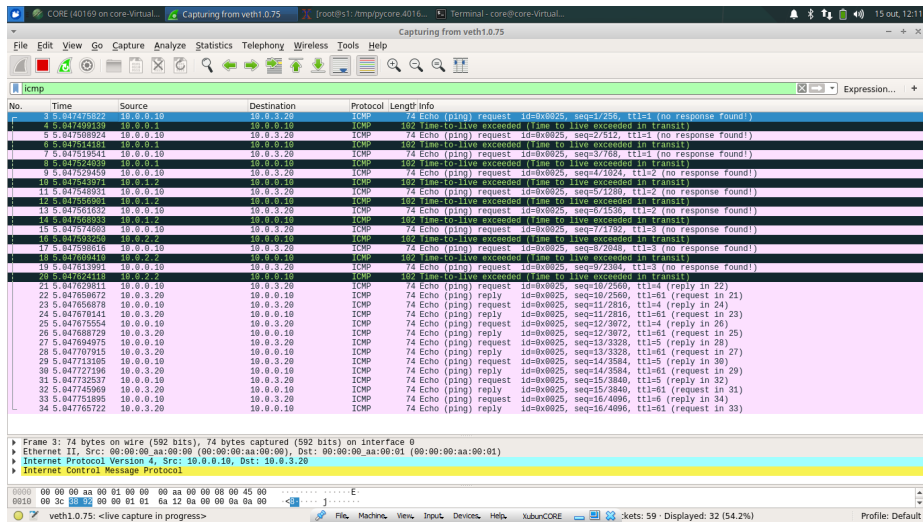


Figura 1.3: Tráfego ICMP enviado e recebido por s1

1.3 Alínea 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 inicial mínimo do campo TTL para alcançar o destino h5 deverá ser 4, uma vez que os pacotes com TTL inferior a 4 recebem uma mensagem de erro, tal como podemos ver na figura 1.3.

1.4 Alínea d

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

Tendo em conta a figura 1.2 concluímos que o Round-Trip-Time é, aproximadamente, 0.126667.

$$RTT = (((0.035 + 0.008 + 0.007)/3) + ((0.017 + 0.011 + 0.010)/3) + ((0.021 + 0.013 + 0.012)/3) + ((0.024 + 0.016 + 0.016)/3)) * 2 \cong 0.126667$$

Capítulo 2

Pergunta 2

Começamos por fazer traceroute usando o programa pingplotter no windows.

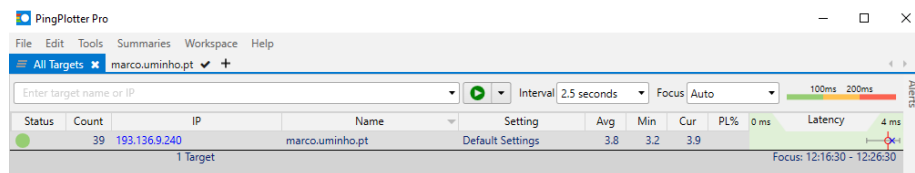


Figura 2.1: Traceroute feito a marco.uminho.pt com o pingplotter

Como o programa referido está constantemente a fazer traceroute decidimos repetir e fazer numa distribuição linux.

```
wtv@wtv-pc /home/wtv ➤ sudo traceroute -I marco.uminho.pt
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 60 byte packets
 1 gw.sa.di.uminho.pt (192.168.100.254)  0.312 ms  0.309 ms  0.307 ms
 2 cisco.di.uminho.pt (193.136.19.254)  0.946 ms  1.261 ms  1.552 ms
 3 marco.uminho.pt (193.136.9.240)  0.920 ms  0.924 ms  0.924 ms
```

Figura 2.2: Traceroute feito a marco.uminho.pt para tamanho padrão

De seguida, capturamos o tráfego usando o wireshark, selecionando a primeira mensagem ICMP capturada e centrando a análise no nível protocolar IP.

No.	Time	Source	Destination	Protocol	Length	Info
10	6.833579965	192.168.100.200	192.168.100.254	DNS	75	Standard query 0x37f6 A marco.uninho.pt
11	6.833984383	192.168.100.200	192.168.100.254	DNS	75	Standard query 0xd1fc AAAA marco.uninho.pt
12	6.834141790	192.168.100.254	192.168.100.200	DNS	129	Standard query response 0xd1fc AAAA marco.uninho.pt 50A dns.uninho.pt
13	6.835513126	192.168.100.254	192.168.100.200	DNS	640	Standard query response 0x37f6 A marco.uninho.pt A 193.136.9.240 NS f.dns.pt NS g.dns.pt NS ns.dns.br
14	6.835559583	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=2/512, ttl=1 (no response found)
15	6.835594868	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=3/768, ttl=1 (no response found)
16	6.83598185	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=4/1024, ttl=1 (no response found)
17	6.835987787	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=5/1280, ttl=2 (no response found)
18	6.836005810	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=6/1536, ttl=2 (no response found)
19	6.836110881	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=7/1792, ttl=3 (reply in 34)
20	6.836117215	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=8/2048, ttl=3 (reply in 35)
21	6.836119091	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=9/2304, ttl=3 (reply in 36)
22	6.83622945	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=10/2560, ttl=4 (reply in 38)
23	6.83625637	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=11/2816, ttl=4 (reply in 39)
24	6.836277859	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=12/3072, ttl=4 (reply in 40)
25	6.836303260	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=13/3328, ttl=5 (reply in 41)
26	6.836334481	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=14/3584, ttl=5 (reply in 42)
27	6.836359961	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=15/3840, ttl=5 (reply in 43)
28	6.836363207	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=16/4096, ttl=5 (reply in 44)

Figura 2.3: Tráfego capturado pelo wireshark

2.1 Alínea a

Qual é o endereço IP da interface ativado seu computador?

O endereço IP da interface ativado no nosso computador é 198.168.100.200, tal como podemos ver no campo Source da figura 2.2.

2.2 Alínea b

Qual é o valor do campo protocolo? O que identifica?

O valor do campo protocolo é 1 e identifica o protocolo ICMP, tal como podemos ver no campo Protocol da figura 2.2.

2.3 Alínea 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 IP tem 20 bytes, como se pode observar no campo Header Length da figura 2.2. Para calcular o tamanho do payload, subtraímos ao número total de bytes o tamanho do cabeçalho, ou seja, $60 - 20 = 40$ bytes.

2.4 Alínea d

O datagrama IP foi fragmentado? Justifique.

Na figura 2.3, observamos que no campo Flags, o Fragment offset é 0. Assim, se existirem mais fragmentos este será o primeiro. Na flag More fragments,

podemos verificar se existem mais fragmentos para além do atual. Se o seu valor for 1, existem; se for 0, não existem. Como estamos no primeiro fragmento e não existem mais (o valor de More Fragments é 0), podemos concluir que este é o datagrama original.

```

▼ 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 2.4: Campo Flags

2.5 Alínea e

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

No.	Time	Source	Destination	Protocol	Length	Info
14	0.635589588	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=1/256, ttl=1 (no response found)
15	0.635594688	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=2/512, ttl=1 (no response found)
16	0.635598185	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=3/768, ttl=1 (no response found)
17	0.635602707	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=4/1024, ttl=2 (no response found)
18	0.635605918	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=5/1280, ttl=2 (no response found)
19	0.635618881	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=6/1536, ttl=2 (no response found)
20	0.635617215	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=7/1792, ttl=3 (reply in 34)
21	0.635619091	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=8/2048, ttl=3 (reply in 35)
22	0.635622045	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=9/2304, ttl=3 (reply in 36)
23	0.635625637	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=10/2560, ttl=4 (reply in 38)
24	0.635627969	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=11/2816, ttl=4 (reply in 39)
25	0.635630268	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=12/3072, ttl=4 (reply in 40)
26	0.635632481	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=13/3328, ttl=5 (reply in 41)
27	0.635635961	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=14/3584, ttl=5 (reply in 42)
28	0.635639297	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=15/3840, ttl=5 (reply in 43)
29	0.635641347	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=16/4096, ttl=6 (reply in 44)
47	0.635912293	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=17/4352, ttl=6 (reply in 51)
48	0.635923128	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=18/4608, ttl=6 (reply in 52)
49	0.635926954	192.168.100.200	193.136.9.240	ICMP	74	Echo (ping) request id=0x1443, seq=19/4864, ttl=7 (reply in 53)
50	0.635930985	192.168.100.254	192.168.100.200	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
51	0.635932190	192.168.100.254	192.168.100.200	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
52	0.635946248	193.136.19.254	192.168.100.200	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
53	0.635948676	193.136.19.254	192.168.100.200	ICMP	74	Time-to-live exceeded (Time to live exceeded in transit)

Frame 34: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
 Ethernet II, Src: HewlettP e6:53:f6 (3c:52:82:e6:53:f6), Dst: Vmware d2:19:f9 (00:0c:29:d2:19:f9)
 Internet Protocol Version 4, Src: 192.168.100.200, Dst: 193.136.9.240
 ICMP, Type: Echo (ping) request, Seq: 1/256, TTL: 1, ID: 0x1443, Length: 74
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 .. Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 68
 Identification: 0xa5e7 (42471)
 Flags: 0x0000
 Time to Live: 1
 Protocol: ICMP (1)
 Header checksum: 0x2f1 [validation disabled]
 Header checksum status: Unverified
 Source: 192.168.100.200
 Destination: 193.136.9.240
 Internet Control Message Protocol
 0000 00 0c 29 d2 19 f9 3c 52 82 e6 53 f6 08 00 05 06 ... <R S
 Internet Protocol Version 4 (ip), 20 bytes

Figura 2.5: Pacotes ordenados de acordo com o endereço de IP fonte

Os campos que variam no cabeçalho IP de pacote para pacote são o identificador de pacote (*Identification*) e o TTL como podemos observar na figura 2.5.

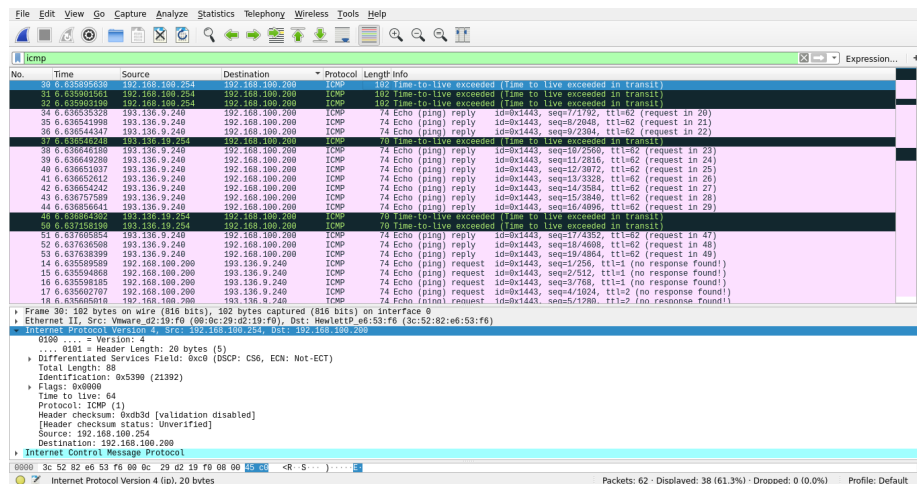
2.6 Alínea f

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

Observamos que o valor do identificador de pacote incrementa 1 a cada linha e o TTL incrementa 1 de 4 em 4 linhas.

2.7 Alínea 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?



The image shows a Wireshark packet capture of ICMP messages. The packets are sorted by destination IP address. The table below represents the data visible in the packet list pane.

No.	Time	Source	Destination	Protocol	Length	Info
10	0.636909050	192.168.100.254	192.168.100.200	ICMP	74	74 Echo (ping) request - id=0x1443, seq=8/2048, ttl=62 (request in 20)
31	0.636909156	192.168.100.254	192.168.100.200	ICMP	102	Time-to-live exceeded (time to live exceeded in transit)
32	0.636909190	192.168.100.254	192.168.100.200	ICMP	102	Time-to-live exceeded (time to live exceeded in transit)
34	0.636909302	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=11/2816, ttl=62 (request in 24)
35	0.636909390	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=8/2048, ttl=62 (request in 21)
36	0.636909497	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=9/2304, ttl=62 (request in 22)
37	0.636909429	193.136.9.240	192.168.100.200	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
38	0.636909436	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=19/2560, ttl=62 (request in 23)
39	0.636909428	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=11/2816, ttl=62 (request in 24)
40	0.636909307	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=12/3072, ttl=62 (request in 25)
41	0.636909312	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=13/3328, ttl=62 (request in 26)
42	0.636909424	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=14/3584, ttl=62 (request in 27)
43	0.636909750	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=15/3840, ttl=62 (request in 28)
44	0.636909564	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=16/4096, ttl=62 (request in 29)
46	0.636909262	193.136.10.254	192.168.100.200	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
49	0.637551190	193.136.10.254	192.168.100.200	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
51	0.637698854	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=17/4352, ttl=62 (request in 47)
52	0.637698960	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=19/4608, ttl=62 (request in 48)
53	0.637698939	193.136.9.240	192.168.100.200	ICMP	74	74 Echo (ping) reply - id=0x1443, seq=19/4864, ttl=62 (request in 49)
14	0.635598980	192.168.100.200	193.136.9.240	ICMP	74	74 Echo (ping) request - id=0x1443, seq=1/256, ttl=1 (no response found)
15	0.635598980	192.168.100.200	193.136.9.240	ICMP	74	74 Echo (ping) request - id=0x1443, seq=2/512, ttl=1 (no response found)
16	0.635598985	192.168.100.200	193.136.9.240	ICMP	74	74 Echo (ping) request - id=0x1443, seq=3/768, ttl=1 (no response found)
17	0.635600797	192.168.100.200	193.136.9.240	ICMP	74	74 Echo (ping) request - id=0x1443, seq=4/1024, ttl=2 (no response found)
18	0.635600810	192.168.100.200	193.136.9.240	ICMP	74	74 Echo (ping) request - id=0x1443, seq=5/1790, ttl=2 (no response found)

Below the packet list, the packet details pane shows the structure of an ICMP Echo (ping) request. The 'Internet Protocol Version 4' section is expanded, showing the source and destination IP addresses. The 'Internet Control Message Protocol' section is also expanded, showing the type (Echo), code (0), and checksum.

Figura 2.6: Pacotes ordenados de acordo com o endereço de IP destino

O valor de TTL é constante e é de 64. Isto mostra que o pacote está em routing loop, como se pode ver na figura 2.6.

Capítulo 3

Pergunta 3

Fizemos, então, o traceroute para *marco.uminho.pt* mas, desta vez, com tamanho de 4201, já que somos o grupo 1 do nosso turno.

```
wtv@wtv-pc /home/wtv sudo traceroute -I marco.uminho.pt 4201
[sudo] password for wtv:
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 4201 byte packets
 1 gw.sa.di.uminho.pt (192.168.100.254) 0.951 ms 1.144 ms 1.456 ms
 2 cisco.di.uminho.pt (193.136.19.254) 2.879 ms 3.188 ms 3.908 ms
 3 marco.uminho.pt (193.136.9.240) 4.634 ms 4.944 ms 5.249 ms
```

Figura 3.1: Tráfego capturado pelo wireshark para um tamanho de pacote de 4201

Capturamos o tráfego gerado com ajuda do *Wireshark*.

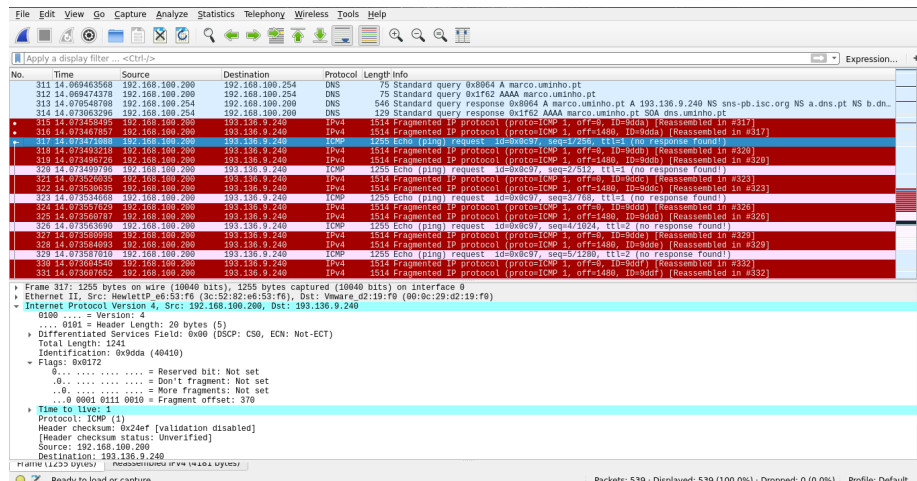


Figura 3.2: Traceroute feito a marco.uminho.pt para tamanho padrão

3.1 Alínea a

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

O tamanho de pacote máximo é de 1500 bytes. Como o tamanho que usamos foi de 4201 bytes, teve necessariamente de fragmentar, pois é demasiado grande para circular na rede.

3.2 Alínea 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?

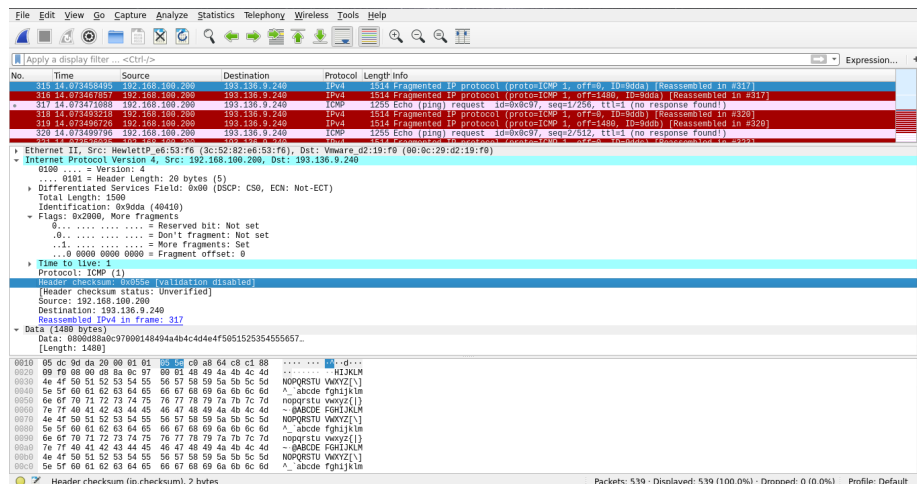


Figura 3.3: Primeiro fragmento do datagrama IP segmentado

No campo *flags* do cabeçalho, podemos observar que o datagrama tem mais fragmentos (*more fragments*). Trata-se do primeiro fragmento, pois tem 0 como *fragment offset*. O tamanho do datagrama é 1480 bytes, pois temos 20 bytes de header ($1500 - 20 = 1480$).

3.3 Alínea c

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

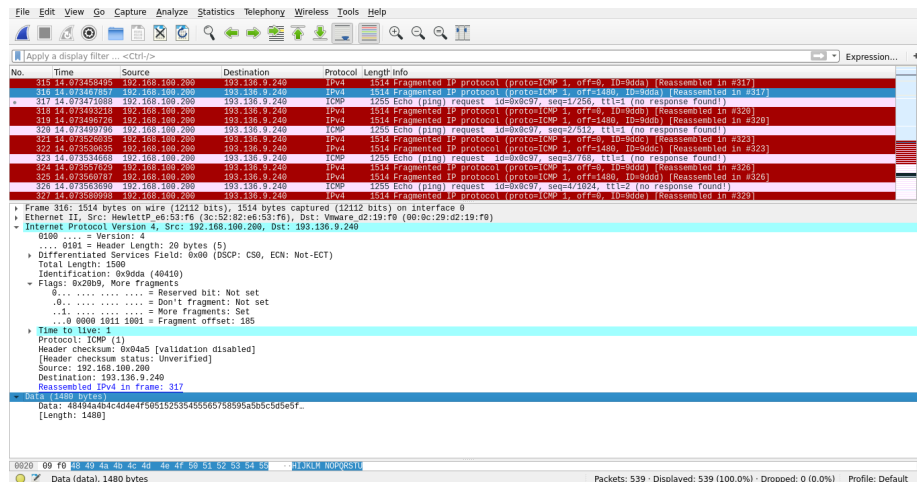


Figura 3.4: Segundo fragmento do datagrama IP segmentado

Não se trata do primeiro fragmento, pois tem como *fragment offset* um valor diferente de 0. O campo *more fragments* tem como valor *set* e, assim, concluímos que tem mais fragmentos.

3.4 Alínea d

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

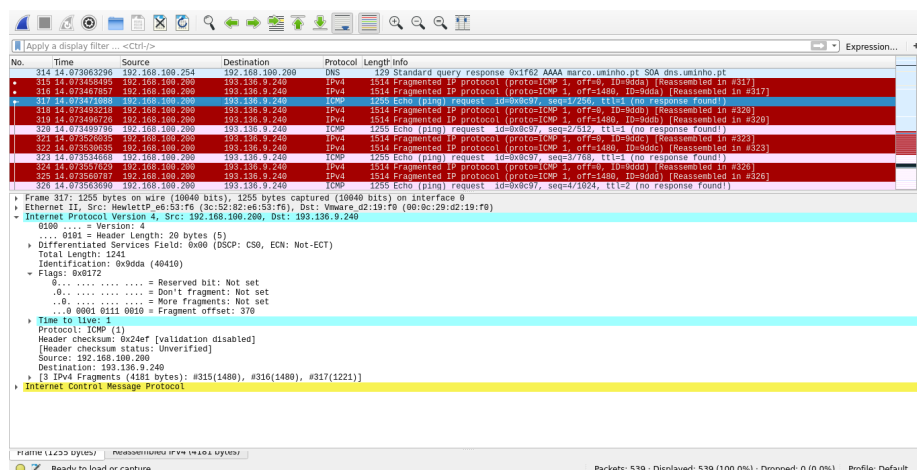


Figura 3.5: Terceiro fragmento do datagrama IP segmentado

Como podemos verificar, temos a mesma identificação dos 3 fragmentos das imagens 3.3, 3.4 e 3.5 (0x9dda), o que nos permite concluir que se tratam do

mesmo datagrama original. Sabemos, ainda, que o fragmento da imagem 3.5 (3º Fragmento) é o último, já que a flag *more fragments* é 0.

3.5 Alínea 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.

Entre os diferentes fragmentos (das imagens 3.3, 3.4 e 3.5), os campos que mudam no cabeçalho IP são as *flags*:

- **Fragment offset** que indica a posição do fragmento no datagrama original.
- **More Fragments** que indica se existem, ou não, mais fragmentos.

A partir das flags *fragment offset* podemos ordenar os fragmentos por ordem crescente e, assim, recriar o datagrama original.

Parte II

Capítulo 1

Pergunta 1

1.1 Alínea 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.

Na figura abaixo, é possível observar os vários endereços IP de cada equipamento. Também podemos verificar que a máscara de rede é 255.255.255.0, uma vez que, na notação CIDR, o número de bits é 24 (/24).

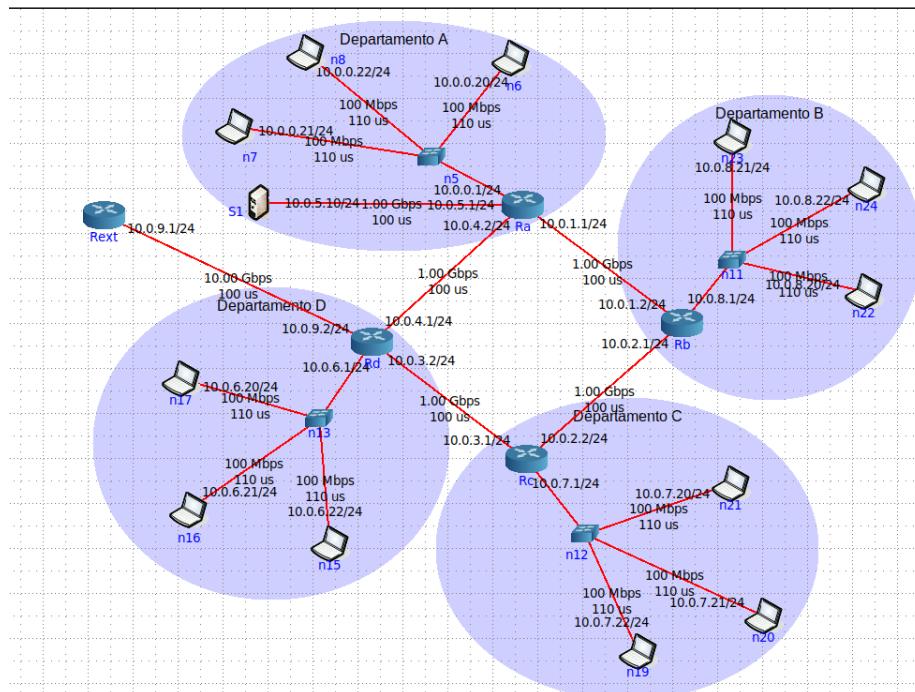


Figura 1.1: Topologia da organização MIEI-RC

1.2 Alínea b

Trata-se de endereços públicos ou privados? Porquê

Os intervalos de endereços privados são:

- 10.0.0.0 a 10.255.255.255 (10.0.0.0 /8)
- 172.16.0.0 a 172.31.255.255 (172.16.0.0 /12)
- 192.168.0.0 a 192.168.255.255 (192.168.0.0 /16)

Uma vez que todos os endereços de rede pertencem ao primeiro intervalo, podemos concluir que se tratam de endereços privados.

1.3 Alínea c

Por que razão não é atribuído um endereço IP aos *switches*?

Os *switches* intervêm na camada de ligação 2, sendo transparentes à camada de ligação 3, onde se encontram os endereços IP. Assim, não será necessário atribuir-lhes um endereço, uma vez que estes encaminham os pacotes, tendo em conta apenas os endereços MAC do equipamento.

1.4 Alínea 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).

Para verificar se existia conectividade IP entre os laptops dos vários departamentos e o servidor do departamento A, recorreremos ao comando *ping* que envia pacotes para o último. Quando este os recebe, envia uma mensagem de resposta, que contém o número do pacote e o tempo de ida e volta. Caso tal não se verifique, envia uma mensagem de erro. Nas figuras abaixo, podemos observar que todos os pacotes foram entregues e, assim, concluir que existe conectividade entre cada um dos laptops de cada departamento e o servidor do departamento A.

```
root@S1: /tmp/pycore.43677/S1.conf
root@S1:/tmp/pycore.43677/S1.conf# ping 10.0.0.21
PING 10.0.0.21 (10.0.0.21) 56(84) bytes of data.
64 bytes from 10.0.0.21: icmp_seq=1 ttl=63 time=2.16 ms
64 bytes from 10.0.0.21: icmp_seq=2 ttl=63 time=0.950 ms
64 bytes from 10.0.0.21: icmp_seq=3 ttl=63 time=0.968 ms
^C
--- 10.0.0.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2031ms
rtt min/avg/max/mdev = 0.950/1.362/2.168/0.569 ms
root@S1:/tmp/pycore.43677/S1.conf# ping 10.0.0.22
PING 10.0.0.22 (10.0.0.22) 56(84) bytes of data.
64 bytes from 10.0.0.22: icmp_seq=1 ttl=63 time=1.40 ms
64 bytes from 10.0.0.22: icmp_seq=2 ttl=63 time=1.34 ms
64 bytes from 10.0.0.22: icmp_seq=3 ttl=63 time=1.37 ms
^C
--- 10.0.0.22 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.343/1.373/1.400/0.048 ms
root@S1:/tmp/pycore.43677/S1.conf# ping 10.0.0.20
PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.
64 bytes from 10.0.0.20: icmp_seq=1 ttl=63 time=9.25 ms
64 bytes from 10.0.0.20: icmp_seq=2 ttl=63 time=51.3 ms
64 bytes from 10.0.0.20: icmp_seq=3 ttl=63 time=41.7 ms
^C
--- 10.0.0.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2012ms
rtt min/avg/max/mdev = 9.256/34.098/51.309/17.997 ms
```

Figura 1.2: Conectividade entre o laptop do departamento A e o servidor

```
root@S1: /tmp/pycore.43677/S1.conf
root@S1:/tmp/pycore.43677/S1.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=17.1 ms
64 bytes from 10.0.8.20: icmp_seq=2 ttl=62 time=47.8 ms
64 bytes from 10.0.8.20: icmp_seq=3 ttl=62 time=6.57 ms
^C
--- 10.0.8.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 6.579/23.861/47.826/17.490 ms
root@S1:/tmp/pycore.43677/S1.conf# ping 10.0.8.21
PING 10.0.8.21 (10.0.8.21) 56(84) bytes of data.
64 bytes from 10.0.8.21: icmp_seq=1 ttl=62 time=2.08 ms
64 bytes from 10.0.8.21: icmp_seq=2 ttl=62 time=1.24 ms
64 bytes from 10.0.8.21: icmp_seq=3 ttl=62 time=1.21 ms
^C
--- 10.0.8.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.217/1.516/2.083/0.401 ms
root@S1:/tmp/pycore.43677/S1.conf# ping 10.0.8.22
PING 10.0.8.22 (10.0.8.22) 56(84) bytes of data.
64 bytes from 10.0.8.22: icmp_seq=1 ttl=62 time=13.6 ms
64 bytes from 10.0.8.22: icmp_seq=2 ttl=62 time=26.1 ms
64 bytes from 10.0.8.22: icmp_seq=3 ttl=62 time=45.0 ms
^C
--- 10.0.8.22 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 13.670/28.290/45.098/12.923 ms
```

Figura 1.3: Conectividade entre o laptop do departamento B e o servidor

```
root@S1: /tmp/pycore.43677/S1.conf
root@S1: /tmp/pycore.43677/S1.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=2,22 ms
64 bytes from 10.0.7.20: icmp_seq=2 ttl=61 time=3,02 ms
64 bytes from 10.0.7.20: icmp_seq=3 ttl=61 time=2,17 ms
^C
--- 10.0.7.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 2.172/2,476/3,029/0,395 ms
root@S1: /tmp/pycore.43677/S1.conf# ping 10.0.7.21
PING 10.0.7.21 (10.0.7.21) 56(84) bytes of data.
64 bytes from 10.0.7.21: icmp_seq=1 ttl=61 time=17,8 ms
64 bytes from 10.0.7.21: icmp_seq=2 ttl=61 time=124 ms
64 bytes from 10.0.7.21: icmp_seq=3 ttl=61 time=126 ms
^C
--- 10.0.7.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 17,853/89,645/126,417/50,771 ms
root@S1: /tmp/pycore.43677/S1.conf# ping 10.0.7.22
PING 10.0.7.22 (10.0.7.22) 56(84) bytes of data.
64 bytes from 10.0.7.22: icmp_seq=1 ttl=61 time=2,45 ms
64 bytes from 10.0.7.22: icmp_seq=2 ttl=61 time=2,01 ms
64 bytes from 10.0.7.22: icmp_seq=3 ttl=61 time=3,10 ms
^C
--- 10.0.7.22 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 2,019/2,527/3,105/0,447 ms
```

Figura 1.4: Conectividade entre o laptop do departamento C e o servidor

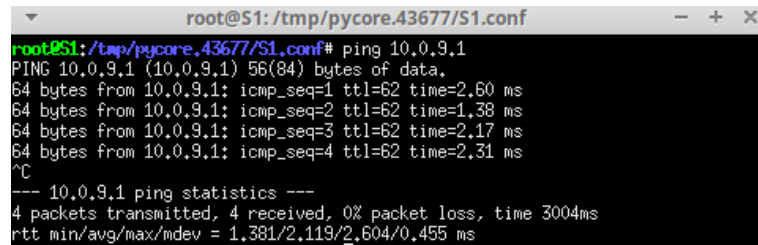
```
root@S1: /tmp/pycore.43677/S1.conf
root@S1: /tmp/pycore.43677/S1.conf# ping 10.0.6.20
PING 10.0.6.20 (10.0.6.20) 56(84) bytes of data.
64 bytes from 10.0.6.20: icmp_seq=1 ttl=62 time=11,1 ms
64 bytes from 10.0.6.20: icmp_seq=2 ttl=62 time=7,95 ms
64 bytes from 10.0.6.20: icmp_seq=3 ttl=62 time=18,9 ms
^C
--- 10.0.6.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2009ms
rtt min/avg/max/mdev = 7,955/12,673/18,965/4,631 ms
root@S1: /tmp/pycore.43677/S1.conf# ping 10.0.6.21
PING 10.0.6.21 (10.0.6.21) 56(84) bytes of data.
64 bytes from 10.0.6.21: icmp_seq=1 ttl=62 time=2,36 ms
64 bytes from 10.0.6.21: icmp_seq=2 ttl=62 time=1,20 ms
64 bytes from 10.0.6.21: icmp_seq=3 ttl=62 time=0,849 ms
^C
--- 10.0.6.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0,849/1,474/2,367/0,649 ms
root@S1: /tmp/pycore.43677/S1.conf# ping 10.0.6.22
PING 10.0.6.22 (10.0.6.22) 56(84) bytes of data.
64 bytes from 10.0.6.22: icmp_seq=1 ttl=62 time=31,7 ms
64 bytes from 10.0.6.22: icmp_seq=2 ttl=62 time=41,9 ms
64 bytes from 10.0.6.22: icmp_seq=3 ttl=62 time=50,7 ms
^C
--- 10.0.6.22 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 31,770/41,474/50,744/7,753 ms
```

Figura 1.5: Conectividade entre o laptop do departamento D e o servidor

1.5 Alínea e

Verifique se existe conectividade IP do *router* de acesso Rext para o servidor S1.

Adotando o mesmo raciocínio da alínea anterior, concluímos que existe conectividade IP do *router* de acesso Rext para o servidor S1, tal como podemos ver na figura abaixo.

A terminal window titled 'root@S1: /tmp/pycore.43677/S1.conf' displays the output of a 'ping 10.0.9.1' command. The output shows four successful ICMP echo requests with varying response times. The statistics at the bottom indicate 4 packets transmitted, 4 received, 0% packet loss, and a total time of 3004ms.

```
root@S1: /tmp/pycore.43677/S1.conf# ping 10.0.9.1
PING 10.0.9.1 (10.0.9.1) 56(84) bytes of data:
64 bytes from 10.0.9.1: icmp_seq=1 ttl=62 time=2.60 ms
64 bytes from 10.0.9.1: icmp_seq=2 ttl=62 time=1.38 ms
64 bytes from 10.0.9.1: icmp_seq=3 ttl=62 time=2.17 ms
64 bytes from 10.0.9.1: icmp_seq=4 ttl=62 time=2.31 ms
^C
--- 10.0.9.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.381/2.119/2.604/0.455 ms
```

Figura 1.6: Conectividade entre o router Rext e o servidor S1

Capítulo 2

Pergunta 2

Para o router e um laptop do departamento B:

2.1 Alínea 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`).

Através da observação das tabelas de encaminhamento abaixo, podemos inferir a rota que o pacote irá realizar. A coluna "Destination" indica a sub-rede destino, a coluna "Gateway" indica o equipamento pelo qual o pacote irá passar e a coluna "Genmask" indica o tipo de máscara que será usada.

No que toca ao laptop n23 existem duas hipóteses:

- Independentemente do endereço de destino, o pacote será encaminhado para o router B
- Caso o pacote tiver o endereço da sub-rede do departamento B, pode optar por qualquer destino

Quanto ao router Rb existem três hipóteses:

- Quando a coluna "Gateway" tem valor 0.0.0.0, o pacote pode optar por qualquer destino
- Os pacotes que tenham como destino um equipamento da sub-rede 10.0.3.0 ou 10.0.7.0 terão de passar por Rc (10.0.2.2)
- Os pacotes que tenham como destino um equipamento da sub-rede 10.0.0.0, 10.0.4.0, 10.0.5.0, 10.0.6.0 ou 10.0.9.0 terão de passar por Ra (10.0.1.1).


```

root@Rb: /tmp/pycore.38767/Rb.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.1.0          0.0.0.0         255.255.255.0   U          0 0          0 eth0
10.0.2.0          0.0.0.0         255.255.255.0   U          0 0          0 eth1
10.0.3.0          10.0.2.2        255.255.255.0   UG        0 0          0 eth1
10.0.4.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.5.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.6.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0
10.0.7.0          10.0.2.2        255.255.255.0   UG        0 0          0 eth1
10.0.8.0          0.0.0.0         255.255.255.0   U          0 0          0 eth2
10.0.9.0          10.0.1.1        255.255.255.0   UG        0 0          0 eth0

```

Figura 2.1: Tabela de encaminhamento de Rb

```

root@n23: /tmp/pycore.38767/n23.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0          10.0.8.1        0.0.0.0         UG        0 0          0 eth0
10.0.8.0          0.0.0.0         255.255.255.0   U          0 0          0 eth0

```

Figura 2.2: Tabela de encaminhamento de n23

2.2 Alínea b

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

Na figura abaixo, podemos observar que o protocolo OSPF (Open Shortest Path First) está a ser utilizado e, portanto, concluímos que está a ser usado encaminhamento dinâmico.

```

root@Rb: /tmp/pycore.38767/Rb.conf# ps -aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  10572  1696 ?        S      11:50   0:00 /usr/local/bin/
quagga    59  0.0  0.1  27548  3116 ?        Ss     11:50   0:00 /usr/sbin/zebra
quagga    65  0.0  0.1  27344  2832 ?        Ss     11:50   0:00 /usr/sbin/ospf6
quagga    69  0.0  0.1  29808  2928 ?        Ss     11:50   0:00 /usr/sbin/ospfd
root     109  0.0  0.2  30280  5044 pts/4    Ss+    11:52   0:00 /bin/bash
root     118  0.2  0.2  30280  5220 pts/8    Ss     11:54   0:00 /bin/bash
root     127  0.0  0.1  47236  3596 pts/8    R+     11:54   0:00 ps -aux

```

Figura 2.3: Protocolos usados

2.3 Alínea 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.

Nas figuras abaixo, podemos constatar que, ao testar a conectividade entre os diferentes laptops e o servidor S1, nenhum tem ligação com o mesmo, uma

vez que são enviados 28 pacotes e nenhum é recebido, o que resulta de termos eliminado a rota por defeito do servidor. Apenas Ra, que está na mesma rede local que o servidor, tem conectividade com o mesmo.

```
root@Ra: /tmp/pycore.38767/Ra.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=0.856 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=0.271 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=64 time=1.09 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=64 time=0.345 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=64 time=0.256 ms
64 bytes from 10.0.5.10: icmp_seq=6 ttl=64 time=0.272 ms
64 bytes from 10.0.5.10: icmp_seq=7 ttl=64 time=0.256 ms
64 bytes from 10.0.5.10: icmp_seq=8 ttl=64 time=1.10 ms
^C
--- 10.0.5.10 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7144ms
rtt min/avg/max/mdev = 0.256/0.555/1.100/0.364 ms
```

Figura 2.4: Ping de Ra para S1

```
root@Rb: /tmp/pycore.38767/Rb.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
^C
--- 10.0.5.10 ping statistics ---
28 packets transmitted, 0 received, 100% packet loss, time 27640ms
```

Figura 2.5: Ping de Rb para S1

2.4 Alínea 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.38767/S1.conf# route del default
root@S1:/tmp/pycore.38767/S1.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@S1:/tmp/pycore.38767/S1.conf# route add -net 10.0.0.0 netmask 255.255.255.0
gw 10.0.5.1
root@S1:/tmp/pycore.38767/S1.conf# route add -net 10.0.8.0 netmask 255.255.255.0
gw 10.0.5.1
root@S1:/tmp/pycore.38767/S1.conf# route add -net 10.0.7.0 netmask 255.255.255.0
gw 10.0.5.1
root@S1:/tmp/pycore.38767/S1.conf# route add -net 10.0.6.0 netmask 255.255.255.0
gw 10.0.5.1
root@S1:/tmp/pycore.38767/S1.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask        Flags   MSS Window  irtt Iface
10.0.0.0         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@S1:/tmp/pycore.38767/S1.conf#

```

Figura 2.6: Adicionar novas rotas de S1 para os respectivos departamentos

O comando `route add` indica que os pacotes que tenham como destino a subrede 10.0.0.0 com máscara 24 têm de entrar no gateway 10.0.5.1. Repetimos o processo para os pacotes que tenham o destino 10.0.6.0, 10.0.7.0 e 10.0.8.0. Com o comando `netstat -rn`, verificamos que o servidor S1 está outra vez conectado aos diferentes laptops.

2.5 Alínea 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@n7:/tmp/pycore.38767/n7.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=63 time=8.22 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=63 time=20.3 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=63 time=19.9 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=63 time=3.02 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=63 time=26.0 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 3.021/15.526/26.073/8.531 ms

```

Figura 2.7: Ping de n7 para S1

```

root@n16:/tmp/pycore.38767/n16.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=62 time=1.56 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=2.09 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=3.07 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=2.18 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=1.93 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4005ms
rtt min/avg/max/mdev = 1.560/2.169/3.070/0.501 ms

```

Figura 2.8: Ping de n16 para S1

```

root@n20:/tmp/pycore.38767/n20.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=61 time=3.14 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=1.97 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=2.26 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=61 time=2.38 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=61 time=2.27 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4013ms
rtt min/avg/max/mdev = 1.971/2.405/3.140/0.395 ms

```

Figura 2.9: Ping de n20 para S1

```

root@n24:/tmp/pycore.38767/n24.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=62 time=1.96 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=2.18 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=2.30 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=62 time=1.86 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=62 time=2.09 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 1.862/2.082/2.303/0.161 ms

```

Figura 2.10: Ping de n24 para S1

Como podemos observar nas figuras 3.7, 5.8, 5.9 e 5.10, existe conectividade entre os laptops dos diferentes departamentos e o servidor S1.

Capítulo 3

Pergunta 3

3.0.1 Alínea 1

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.

Sendo que pertencemos ao turno prático PL4 e somos o grupo G01 nesse mesmo turno, dispomos do endereço de rede IP 172.014.32.0/20.

Uma vez que temos 4 departamentos, poderíamos pensar que necessitamos de mais 2 bits, pois $2^2 = 4$, o que resultaria numa nova máscara de 22 bits. Contudo, uma vez que Ra está ligado a um switch e um servidor, concluímos que uma máscara de 22 bits não é suficiente. Então, escolhemos uma máscara de 23 bits que resultará nos seguintes endereços:

- **Departamento A:** 172.014.32.0/23 e 172.014.34.0/23
- **Departamento B:** 172.014.36.0/23
- **Departamento C:** 172.014.38.0/23
- **Departamento D:** 172.014.40.0/23

Relativamente ao departamento A, os vários sistemas envolvidos terão os seguintes endereços:

- **ra** 172.014.32.1/23 e 172.014.34.1/23
- **s1** 172.014.34.10/23
- **n6** 172.014.32.20/23
- **n7** 172.014.32.21/23
- **n8** 172.014.32.22/23

Relativamente ao departamento B, os vários sistemas envolvidos terão os seguintes endereços:

- **rb** 172.014.36.1/23
- **n22** 172.014.36.20/23
- **n23** 172.014.36.21/23
- **n24** 172.014.36.22/23

Relativamente ao departamento C, os vários sistemas envolvidos terão os seguintes endereços:

- **rc** 172.014.38.1/23
- **n19** 172.014.38.20/23
- **n20** 172.014.38.21/23
- **n21** 172.014.38.22/23

Relativamente ao departamento D, os vários sistemas envolvidos terão os seguintes endereços:

- **rd** 172.014.40.1/23
- **n15** 172.014.40.20/23
- **n16** 172.014.40.21/23
- **n17** 172.014.40.22/23

3.0.2 Alínea 2

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

A máscara de rede que usamos foi /23 que, em decimal, corresponde a 255.255.254.0. Assim, restam-nos 9 bits e, portanto, o número de interfaces IP que podemos interligar em cada departamento será $2^9 - 2$ (endereços reservados para rede e broadcast) = 510.

3.0.3 Alínea 3

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

Para verificar que a conectividade IP entre as várias redes locais da organização MIEI-RC é mantida, realizamos o ping entre um laptop do departamento A, o servidor s1 e um laptop de cada departamento. Nas figuras abaixo, verificamos que esta conexão se mantém, uma vez que há comunicação entre os diferentes equipamentos.

```
root@n7: /tmp/pycore.37965/n7.conf
root@n7: /tmp/pycore.37965/n7.conf# ping 172.014.34.10
PING 172.014.34.10 (172.12.34.10) 56(84) bytes of data.
64 bytes from 172.12.34.10: icmp_seq=1 ttl=63 time=1.62 ms
64 bytes from 172.12.34.10: icmp_seq=2 ttl=63 time=0.398 ms
64 bytes from 172.12.34.10: icmp_seq=3 ttl=63 time=0.846 ms
^C
--- 172.014.34.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2038ms
rtt min/avg/max/mdev = 0.398/0.957/1.628/0.508 ms
```

Figura 3.1: Teste de conexão (ping) entre um laptop do departamento A e um servidor do mesmo departamento

```
root@n7: /tmp/pycore.37965/n7.conf
root@n7: /tmp/pycore.37965/n7.conf# ping 172.014.36.20
PING 172.014.36.20 (172.12.36.20) 56(84) bytes of data.
64 bytes from 172.12.36.20: icmp_seq=1 ttl=62 time=1.30 ms
64 bytes from 172.12.36.20: icmp_seq=2 ttl=62 time=0.867 ms
64 bytes from 172.12.36.20: icmp_seq=3 ttl=62 time=1.91 ms
64 bytes from 172.12.36.20: icmp_seq=4 ttl=62 time=1.95 ms
64 bytes from 172.12.36.20: icmp_seq=5 ttl=62 time=2.12 ms
64 bytes from 172.12.36.20: icmp_seq=6 ttl=62 time=1.22 ms
64 bytes from 172.12.36.20: icmp_seq=7 ttl=62 time=1.25 ms
64 bytes from 172.12.36.20: icmp_seq=8 ttl=62 time=2.16 ms
^C
--- 172.014.36.20 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7039ms
rtt min/avg/max/mdev = 0.867/1.600/2.161/0.461 ms
```

Figura 3.2: Teste de conexão (ping) entre um laptop do departamento A e B

```
root@n7: /tmp/pycore.37965/n7.conf
root@n7: /tmp/pycore.37965/n7.conf# ping 172.014.38.20
PING 172.014.38.20 (172.12.38.20) 56(84) bytes of data.
64 bytes from 172.12.38.20: icmp_seq=1 ttl=61 time=15.8 ms
64 bytes from 172.12.38.20: icmp_seq=2 ttl=61 time=14.9 ms
64 bytes from 172.12.38.20: icmp_seq=3 ttl=61 time=16.5 ms
64 bytes from 172.12.38.20: icmp_seq=4 ttl=61 time=20.7 ms
^C
--- 172.014.38.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 14.934/17.026/20.725/2.218 ms
```

Figura 3.3: Teste de conexão (ping) entre um laptop do departamento A e C

```
root@n7: /tmp/pycore.37965/n7.conf
root@n7: /tmp/pycore.37965/n7.conf# ping 172.014.40.20
PING 172.014.40.20 (172.12.40.20) 56(84) bytes of data.
64 bytes from 172.12.40.20: icmp_seq=1 ttl=62 time=6.02 ms
64 bytes from 172.12.40.20: icmp_seq=2 ttl=62 time=3.30 ms
64 bytes from 172.12.40.20: icmp_seq=3 ttl=62 time=81.7 ms
^C
--- 172.014.40.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 3.300/30.349/81.720/36.341 ms
```

Figura 3.4: Teste de conexão (ping) entre um laptop do departamento A e D

Capítulo 4

Conclusões

Parte I

Com a primeira parte deste trabalho prático, conseguimos usar ferramentas como o *Core*, *Wireshark* e *traceroute*, com sucesso. Relativamente ao *Core*, tivemos oportunidade de criar uma rede fictícia (simulação). Com a simulação criada, realizamos o *traceroute* a um host, analisando o tráfego *ICMP* gerado com o auxílio do *Wireshark*. Com o *output* do *traceroute*, conseguimos calcular o RTT e, com o tráfego gerado, conseguimos ver valores como o TTL mínimo para chegar a um determinado destino. Na segunda parte, ainda desta parte 1, fizemos a ligação com *marco.uminho.pt*, enviando pacotes de diferentes tamanhos, usando novamente o *traceroute*. Ao analisar os datagramas IP, verificamos no *Wireshark* que, para pacotes pequenos, não havia fragmentação mas, para pacotes maiores, havia a necessidade de criar vários fragmentos para o conseguir enviar na totalidade.

Parte II

Com a segunda parte deste trabalho prático, percebemos a diferença entre endereços públicos e privados e analisamos tabelas de encaminhamento unicast com o *netstat -rn*. Para além disso, Usamos a ferramenta *ping* para testar a conexão entre equipamentos em diferentes cenários. Distinguimos os diferentes encaminhamentos (estático e dinâmico), observando a existência do protocolo OSPF. Também eliminamos uma rota predefinida, o que nos obrigou a criar novas rotas para que os diferentes departamentos pudessem comunicar com o servidor. Por fim, através de um endereço de rede IP fornecido, tivemos de pôr em prática o *subnetting* para criar endereços suficientes para cada um dos equipamentos.

Posto isto, ao pôr em prática aquilo que foi lecionado nas aulas teóricas, conseguimos consolidar melhor os nossos conhecimentos e acreditamos que os objetivos propostos foram alcançados.