

TP2 Protocolo IPv4

Rafael Silva, José Ramos, and Luís Ferreira

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a74264,a73855,a76936}@alunos.uminho.pt

1 Introduction

O presente relatório destina-se a apresentar um conjunto de perguntas e respostas sobre o protocolo IPv4. O principal objetivo é estudar o IP (Internet Protocol) nas suas principais fileiras, isto é, o estudo do formato de um pacote IP, o endereçamento, o encaminhamento e por fim a fragmentação dos pacotes IP. Na primeira parte do trabalho foi efetuado o estudo sobre o registo dos pacotes IP, enviados e recebidos com o auxílio do programa traceroute, assim como a análise dos vários campos que constituem um datagrama e do seu processo de fragmentação. Na segunda parte continuamos a realização do trabalho com o estudo do endereçamento e encaminhamento IP, onde serão estudadas algumas das técnicas mais relevantes que foram propostas para aumentar a escalabilidade do protocolo IP, mitigar a exaustão dos endereços IPv4 e também reduzir os recursos de memória necessários nos routers para manter as tabelas de encaminhamento. Em suma, este relatório está dividido da mesma maneira que foi feito o processo do trabalho, sendo que começa com um capítulo de Questões e Respostas da parte I e da parte II do trabalho e termina com uma pequena conclusão do grupo face às dificuldades e também apredendizagens sobre o mesmo.

2 Parte 1

2.1 Exercicio 1

Prepare uma topologia CORE para verificar o comportamento do traceroute. Ligue um host (pc) h1 a um router r2; o router r2 a um router r3, que por sua vez, se liga a um host (servidor) s4. (Note que pode não existir conectividade IP imediata entre h1 e s4 até que o routing estabilize). Ajuste o nome dos equipamentos atribuídos por defeito para a topologia do enunciado.

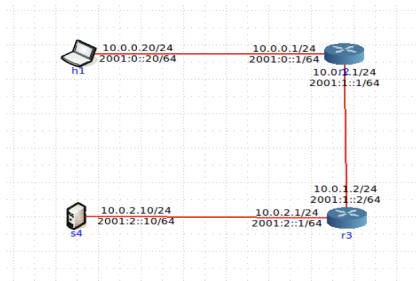


Figura 1. Core

- a) Active o wireshark ou o tcpdump no pc h1. Numa shell de h1, execute o comando traceroute -I para o endereço IP do host s4.

Resposta:

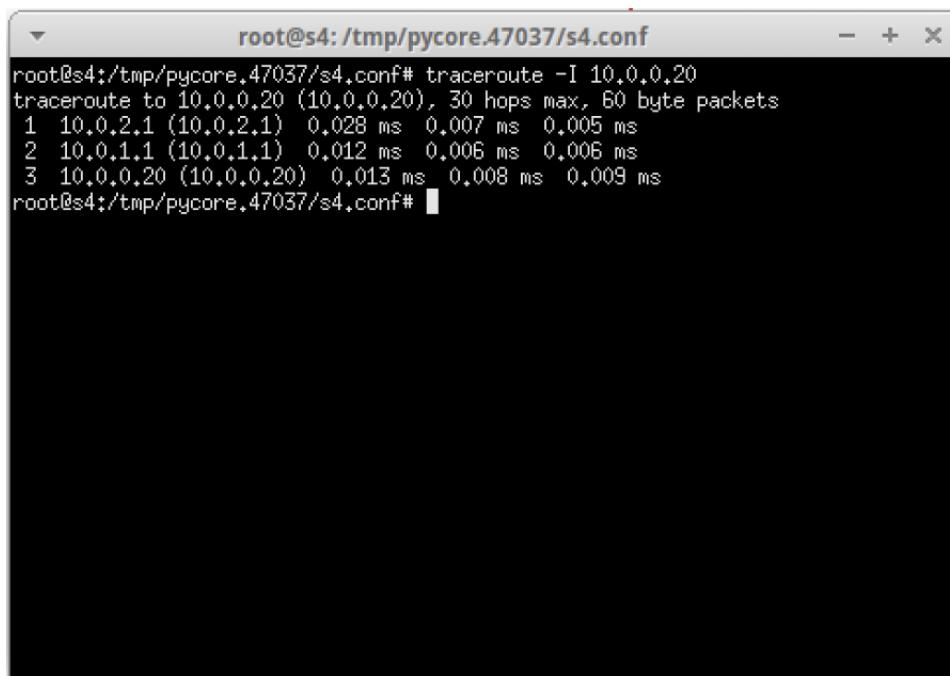
```
root@h1:/tmp/pycore.39592/h1.conf
root@h1:/tmp/pycore.39592/h1.conf# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1  A0 (10.0.0.1)  0.026 ms  0.005 ms  0.004 ms
 2  10.0.1.2 (10.0.1.2)  0.015 ms  0.020 ms  0.008 ms
 3  10.0.2.10 (10.0.2.10)  0.014 ms  0.009 ms  0.008 ms
root@h1:/tmp/pycore.39592/h1.conf#
```

Figura 2. Aplicação do comando traceroute -I para o host s4

b) Registe e analise o tráfego ICMP enviado por h1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

Resposta: O tráfego ICMP enviado por h1 para 10.0.2.10 (s4) corresponde a 3 datagramas com o mesmo TTL de cada vez, pois não existe segurança na rede.

c) Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino s4? Verifique na prática que a sua resposta está correta.



A terminal window titled "root@s4: /tmp/pycore.47037/s4.conf" displays the output of the "traceroute -I 10.0.0.20" command. The output shows three routers along the path to destination 10.0.0.20: 10.0.2.1, 10.0.1.1, and 10.0.0.20. The times for each hop are listed in milliseconds.

```
root@s4:/tmp/pycore.47037/s4.conf# traceroute -I 10.0.0.20
traceroute to 10.0.0.20 (10.0.0.20), 30 hops max, 60 byte packets
1 10.0.2.1 (10.0.2.1) 0.028 ms 0.007 ms 0.005 ms
2 10.0.1.1 (10.0.1.1) 0.012 ms 0.006 ms 0.006 ms
3 10.0.0.20 (10.0.0.20) 0.013 ms 0.008 ms 0.009 ms
root@s4:/tmp/pycore.47037/s4.conf#
```

Figura 3. Aplicação do comando traceroute -I para o host h1

Resposta: O tempo mínimo necessário para alcançar s4 é 3.

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

Resposta:

1. $(0.028 + 0.007 + 0.005) / 3 = 0.01(3) \text{ ms}$
2. $(0.012 + 0.006 + 0.006) / 3 = 0.008 \text{ ms}$
3. $(0.013 + 0.008 + 0.009) / 3 = 0.01 \text{ ms}$

2.2 Exercício 2

No.	Time	Source	Destination	Protocol	Length	Info
28	2.549341	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=1/256, ttl=1 (no response found!)
29	2.558920	172.26.254.254	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
34	2.553126	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=2/512, ttl=1 (no response found!)
35	2.554690	172.26.254.254	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
38	2.554809	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=3/768, ttl=1 (no response found!)
39	2.556326	172.26.254.254	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
41	2.556427	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=4/1024, ttl=2 (no response found!)
43	2.557977	172.16.2.1	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
52	3.563781	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=5/1280, ttl=2 (no response found!)
53	3.565967	172.16.2.1	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
55	3.566442	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=6/1536, ttl=2 (no response found!)
57	3.568078	172.16.2.1	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
60	3.568192	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=7/1792, ttl=3 (no response found!)
61	3.570109	172.16.115.252	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
89	4.575393	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=8/2048, ttl=3 (no response found!)
90	4.579365	172.16.115.252	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
93	4.579471	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=9/2304, ttl=3 (no response found!)
94	4.582715	172.16.115.252	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
97	4.582836	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=10/2560, ttl=4 (reply in 100)
100	4.585663	193.136.9.240	172.26.19.45	ICMP	1514	Echo (ping) reply id=0xa68b, seq=10/2560, ttl=61 (request in 97)
105	4.588723	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=11/2816, ttl=4 (reply in 108)
108	4.591237	193.136.9.240	172.26.19.45	ICMP	1514	Echo (ping) reply id=0xa68b, seq=11/2816, ttl=61 (request in 105)
111	4.591356	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0xa68b, seq=12/3072, ttl=4 (reply in 114)
114	4.594611	193.136.9.240	172.26.19.45	ICMP	1514	Echo (ping) reply id=0xa68b, seq=12/3072, ttl=61 (request in 111)

► Ethernet II, Src: Apple_79:c1:5f (78:4f:43:79:c1:5f), Dst: ComdaEnt_ff:94:00 (00:00:00:ff:94:00)
 ► Internet Protocol Version 4, Src: 172.26.19.45, Dst: 193.136.9.240
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 ► Differentiated Services Field: 0x00 (DSSCP: CS0, ECN: Not-ECT)
 Total Length: 542
 Identification: 0xa68c (42636)
 ▼ Flags: 0x0172
 0... = Reserved bit: Not set
 .0.... = Don't fragment: Not set
 ..0.... = More fragments: Not set
 ...0 0001 0111 0010 = Fragment offset: 370
 ► Time to live: 1
 Protocol: ICMP (1)
 Header checksum: 0x8521 [validation disabled]
 [Header checksum status: Unverified]
 Source: 172.26.19.45
 Destination: 193.136.9.240
 ► [3 IPv4 Fragments (3482 bytes): #26(1480), #27(1480), #28(522)]
 ► Internet Control Message Protocol

Figura 4. Datagrama IP em estudo.

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

Source: 172.26.19.45

Figura 5. Endereço IP.

Resposta: O endereço IP da interface ativa do computador é o endereço “Source”, que tem o valor 172.26.19.45

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

Protocol: ICMP (1)

Figura 6. Campo Protocolo.

Resposta: O valor do campo protocolo é 1 e identifica o tipo de mensagem, neste caso, é uma mensagem ICMP.

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?

```
.... 0101 = Header Length: 20 bytes (5)
▶ Differentiated Services Field: 0x00 (DSCH: CS0, ECN: Not-ECT)
Total Length: 542
```

Figura 7. Valor do campo protocolo.

Resposta: O valor do cabeçalho do datagrama são **20 bytes** e o tamanho total são **542 bytes**, portanto para calcular o tamanho do payload, subtrai-se o valor do cabeçalho ao valor total e encontra-se o tamanho do payload, que neste caso são : **542-20 = 522 bytes**.

d) O datagrama IP foi fragmentado? Justifique.

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

Figura 8. Valor do campo More Fragments e Fragments OffSet.

Resposta: O datagrama IP **foi fragmentado**, porque verificamos que no campo Fragment offset o valor era **370**, portanto, isto quer dizer que existe mais fragmentos deste datagrama, e visto que o campo Fragment OffSet contém um valor diferente de zero, indica-nos que este fragmento não é o único deste datagrama.

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.

Resposta:

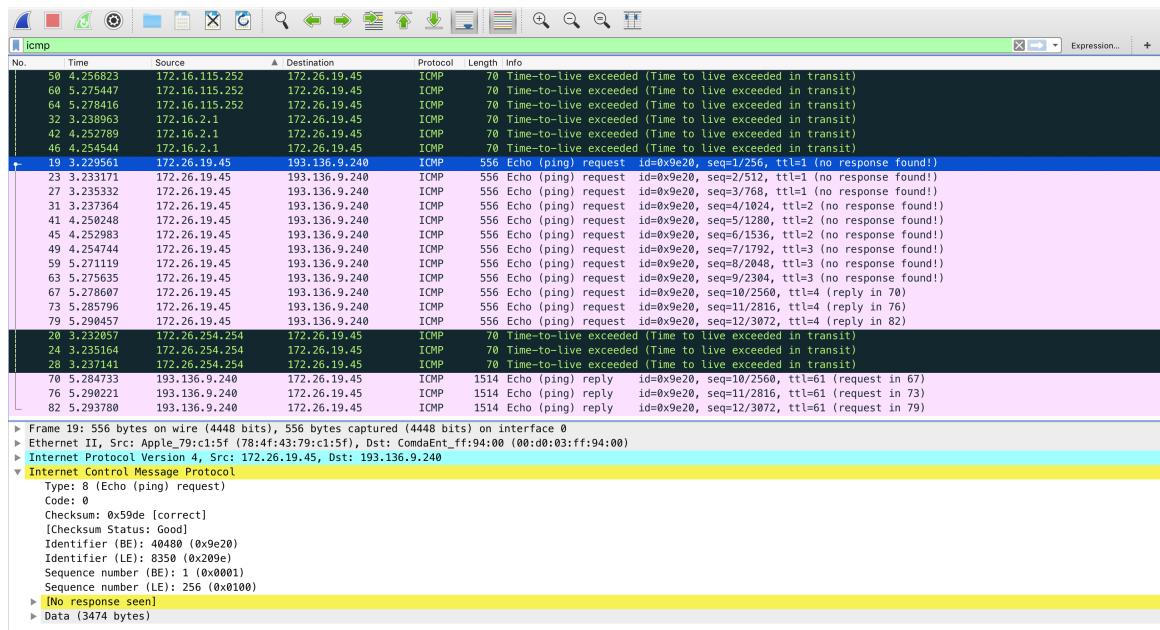


Figura 9. Mensagens ICMP ordenadas pelo campo Source

Sequence number (BE): 1 (0x0001)
 Sequence number (LE): 256 (0x0100)

Figura 10. Mensagem ICMP nº 1, frame 19

Sequence number (BE): 2 (0x0002)
 Sequence number (LE): 512 (0x0200)

Figura 11. Mensagem ICMP nº 2, frame 23

Sequence number (BE): 3 (0x0003)
Sequence number (LE): 768 (0x0300)

Figura 12. Mensagem ICMP nº 3, frame 27

Sequence number (BE): 4 (0x0004)
Sequence number (LE): 1024 (0x0400)

Figura 13. Mensagem ICMP nº 4, frame 31

Sequence number (BE): 5 (0x0005)
Sequence number (LE): 1280 (0x0500)

Figura 14. Mensagem ICMP nº 5, frame 41

Sequence number (BE): 6 (0x0006)
Sequence number (LE): 1536 (0x0600)

Figura 15. Mensagem ICMP nº 6, frame 45

Sequence number (BE): 7 (0x0007)
Sequence number (LE): 1792 (0x0700)

Figura 16. Mensagem ICMP nº 7, frame 49

Sequence number (BE): 8 (0x0008)
Sequence number (LE): 2048 (0x0800)

Figura 17. Mensagem ICMP nº 8, frame 59

Sequence number (BE): 9 (0x0009)
Sequence number (LE): 2304 (0x0900)

Figura 18. Mensagem ICMP nº 9, frame 63

Sequence number (BE): 10 (0x000a)
Sequence number (LE): 2560 (0x0a00)

Figura 19. Mensagem ICMP nº 10, frame 67

Sequence number (BE): 11 (0x000b)
Sequence number (LE): 2816 (0xb000)

Figura 20. Mensagem ICMP nº 11, frame 73

Sequence number (BE): 12 (0x000c)
Sequence number (LE): 3072 (0xc000)

Figura 21. Mensagem ICMP nº 12, frame 79

Nestas imagens podemos ver os doze pacotes enviados pelo nosso computador ate ao host destino, neste caso, três destes pacotes com o TTL = 1, três pacotes com TTL = 2, três pacotes com TTL = 3, que nao conseguiram chegar ao destino e os outros três pacotes com TTL = 4. Cada um destes pacotes tem um sequence number que lhe é atribuido por ordem de saida da origem, portanto a figura 10, 11 e 12 correspondem aos pacotes com TTL = 1, as figuras 13, 14 e 15 correspondem ao TTL = 2, as figuras 16, 17 e 18 correspondem ao TTL = 3 e por fim as figuras 19, 20 e 21 correspondem ao TTL = 4.

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

Resposta: Sim, cada pacote enviado para o host tem um Sequence Number e é através dele que relacionamos os pacotes enviados aos pacotes recebidos dos host destino, desta forma, quando recebemos por exemplo uma mensagem com TTL exceed através do Sequence Number dessa mensagem conseguimos detetar a que pacote corresponde.

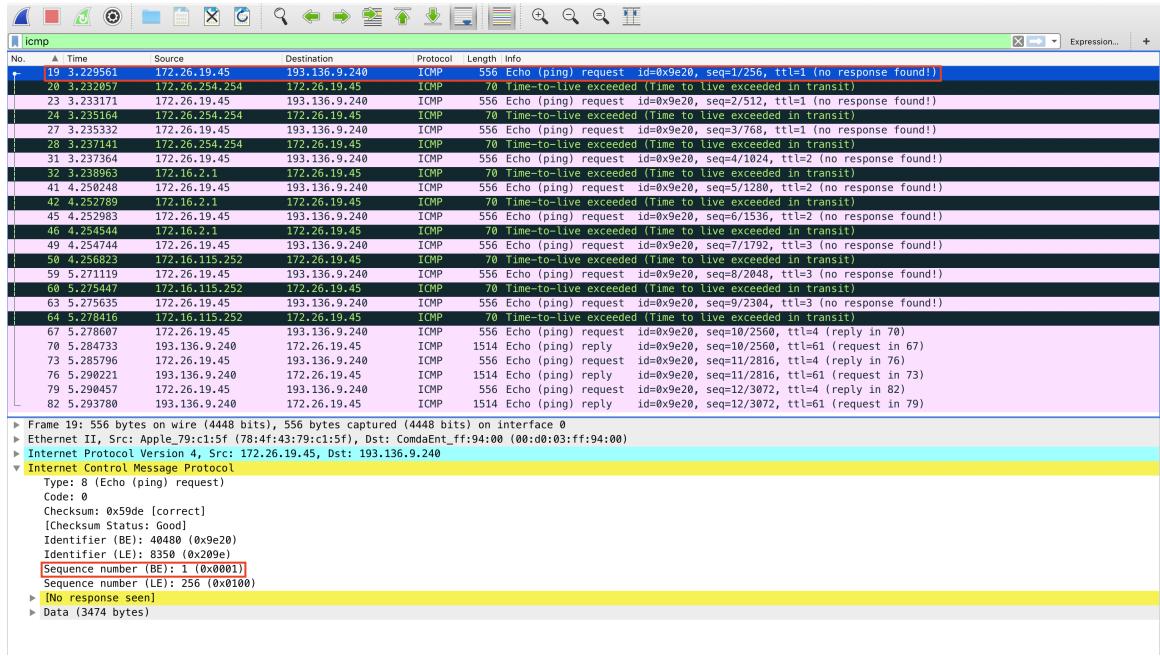


Figura 22. Mensagens ICMP enviada pelo nosso computador com Sequence Number = 1

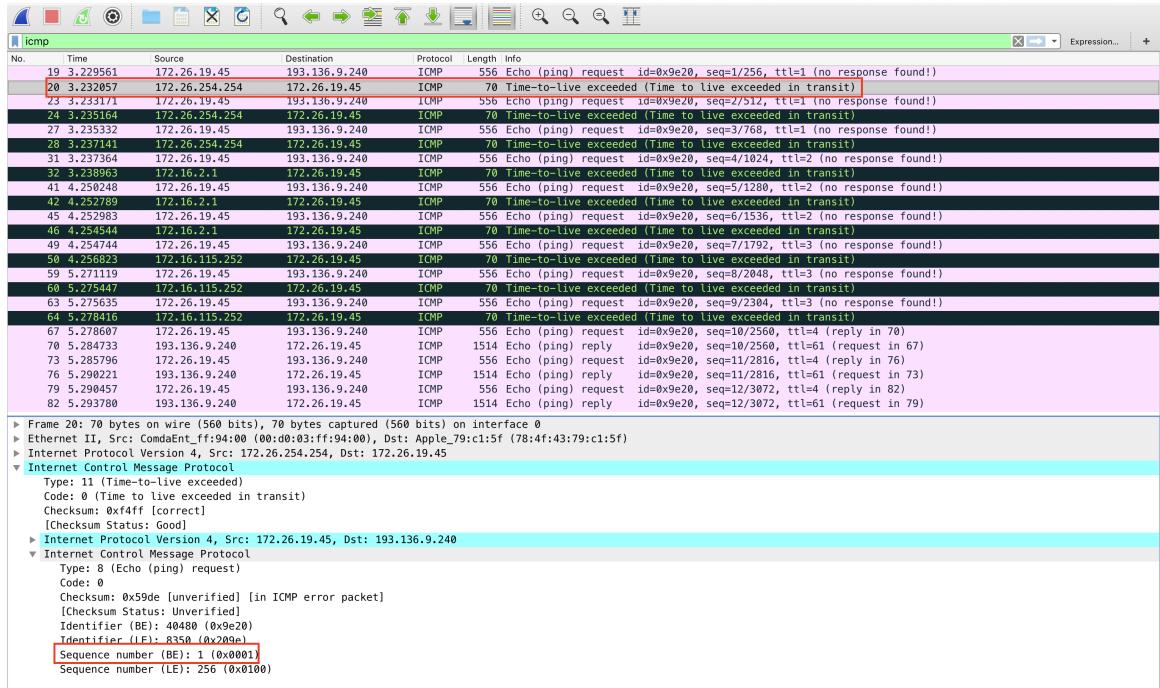


Figura 23. Mensagens ICMP enviada pelo primeiro host em que o pacote passou, com o aviso de TTL exceed e Sequence Number = 1

g) Ordene o tráfego capturado por 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?

Resposta: Sim, porque como conseguimos chegar ao destino com pacotes de TTL = 4, isto quer dizer que os nossos 3 pacotes com TTL = 1, TTL = 2 e TTL = 3 tiveram, todos, respostas com TTL exceed. As respostas tambem tiverem todas um TTL comum, que foi TTL = 253, visto que foram todas mandadas pelo mesmo host, a programação dele foi feita para enviar replies com TTL = 253 e assim foi feito. E por fim, como ja explicamos em cima sao sempre enviados 3 pacotes com o mesmo numero de TTL porque é uma norma do endereçamento IP.

No.	Time	Source	Destination	Protocol	Length	Info
19	3.229561	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=1/256, ttl=1 (no response found!)
20	3.232057	172.26.254.254	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
23	3.233171	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=2/512, ttl=1 (no response found!)
24	3.235164	172.26.254.254	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
27	3.235332	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=3/768, ttl=1 (no response found!)
28	3.237141	172.26.254.254	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
31	3.237364	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=4/1024, ttl=2 (no response found!)
32	3.238963	172.16.2.1	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
41	4.250248	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=5/1280, ttl=2 (no response found!)
42	4.252789	172.16.2.1	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
45	4.252983	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=6/1536, ttl=2 (no response found!)
46	4.254544	172.16.2.1	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
49	4.254744	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=7/1792, ttl=3 (no response found!)
50	4.256823	172.16.115.252	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
59	5.271119	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=8/2048, ttl=3 (no response found!)
60	5.275947	172.16.115.252	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
63	5.275635	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=9/2384, ttl=3 (no response found!)
64	5.278416	172.16.115.252	172.26.19.45	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
67	5.278607	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=10/2560, ttl=4 (reply in 70)
70	5.284733	193.136.9.240	172.26.19.45	ICMP	1514	Echo (ping) reply id=0x9e20, seq=10/2560, ttl=61 (request in 67)
73	5.285796	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=11/2816, ttl=4 (reply in 76)
76	5.290221	193.136.9.240	172.26.19.45	ICMP	1514	Echo (ping) reply id=0x9e20, seq=11/2816, ttl=61 (request in 73)
79	5.290457	172.26.19.45	193.136.9.240	ICMP	556	Echo (ping) request id=0x9e20, seq=12/3072, ttl=4 (reply in 82)
82	5.293780	193.136.9.240	172.26.19.45	ICMP	1514	Echo (ping) reply id=0x9e20, seq=12/3072, ttl=61 (request in 79)

Frame 64: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface 0
 ▶ Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: Apple_79:c1:f5 (78:4f:43:79:c1:f5)
 ▶ Internet Protocol Version 4, Src: 172.16.115.252, Dst: 172.26.19.45
 0100 = Version: 4
 0101 = Header Length: 20 bytes (5)
 ▶ Differentiated Services Field: 0x00 (DSFP: CS0, ECN: Not-ECT)
 Total Length: 56
 Identification: 0x3577 (13687)
 Flags: 0x0000
 Time to live: 253
 Protocol: ICMP (1)
 Header checksum: 0xa8f9 [validation disabled]
 [Header checksum status: Unverified]
 Source: 172.16.115.252
 Destination: 172.26.19.45
 ▶ Internet Control Message Protocol

Figura 24. Valor do TTL das respostas com TTL exceed

2.3 Exercicio 3

Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempo de captura. Observe o tráfego depois do tamanho de pacote ter sido definido para 35XX bytes. (3502)

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

10 0.004051	172.26.254.254	172.26.69.211	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
11 0.005756	172.26.69.211	193.136.9.254	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=88f0) [Reassembled in #13]
12 0.005757	172.26.69.211	193.136.9.254	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=88f0) [Reassembled in #13]
+ 13 0.005757	172.26.69.211	193.136.9.254	ICMP	556 Echo (ping) request id=0x88ee, seq=2/512, ttl=1 (no response found!)
14 0.007724	172.26.254.254	172.26.69.211	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)

Figura 25.

Resposta: Como podemos ver na figura houve necessidade de fragmentar o pacote inicial, pois este era demasiado grande, tendo de se dividir, assumindo um comprimento de 1514 nos dois primeiros segmentos (11 e 12), e no terceiro (13)um comprimento de 556 onde foi “reassembled”.

- 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?

1 0.000000	172.26.69.211	193.136.9.254	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=92)
2 0.000001	172.26.69.211	193.136.9.254	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=92)
+ 3 0.000001	172.26.69.211	193.136.9.254	ICMP	556 Echo (ping) request id=0x920b, seq=1/256, ttl=1 (no response found!)
				[Frame is marked: False] [Frame is ignored: False] [Protocols in frame: eth:ether-type:ip:data] [Coloring Rule Name: TTL low or unexpected] [Coloring Rule String: (! ip.dst == 224.0.0.0/4 && ip.ttl < 5 && ! pim && ! ospf) (ip.dst == 224.0.0.0/24 && ip.dst != 224.0.0.251)]
				Ethernet II, Src: Apple_09:15:06 (80:e6:50:09:15:06), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00) ► Destination: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00) ► Source: Apple_09:15:06 (80:e6:50:09:15:06) Type: IPv4 (0x8000)
				Internet Protocol Version 4, Src: 172.26.69.211, Dst: 193.136.9.254 0100 = Version: 4 0101 = Header Length: 20 bytes (5) ► Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT) Total Length: 1500 Identification: 0x920c (37388) Flags: 0x2000, More fragments 0... = Reserved bit: Not set .0.. = Don't fragment: Not set .1.. = More fragments: Set .0 0000 0000 0000 = Fragment offset: 0
				Time to live: 1 ► [Expert Info (Note/Sequence): "Time To Live" only 1] Protocol: ICMP (1) Header checksum: 0x44a1 [validation disabled] [Header checksum status: Unverified] Source: 172.26.69.211 Destination: 193.136.9.254 Reassembled IPv4 in frame: 3 ► Data (1480 bytes)

Figura 26.

Resposta: “Reassembled IPv4 in frame: 3” indica-nos que é um fragmento que será reconstruído quando receber todos segmentos, isto é na trama 3. “More Fragments: Set” (toma valor 1) indica que existem mais fragmentos para além deste. Trata-se do primeiro fragmento pois o “Fragment offset” é igual a 0. O segundo fragmento vai ter o offset igual a 185 (octetos). Em relação ao tamanho, temos 1480 de dados e 20 de header, logo, no total, 1500.

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?

```

1 0.000000 172.26.69.211 193.136.9.254 IPv4 1514 Fragmented IP protocol [proto=ICMP 1, of=0, Id=92]
2 0.000001 172.26.69.211 193.136.9.254 IPv4 1514 Fragmented IP protocol [proto=ICMP 1, of=1480, Id
+ 3 0.000001 172.26.69.211 193.136.9.254 ICMP 556 Echo (ping) request id=0x920b, seq=1/256, ttl=1 [
  [Frame is marked: False]
  [Frame is ignored: False]
  [Protocols in frame: eth:ether:type:ip:data]
  [Coloring Rule Name: TTL low or unexpected]
  [Coloring Rule String: (! ip.dst == 224.0.0.0/4 && ip.ttl < 5 && ipin && !ospf) || (ip.dst == 224.0.0.24 && ip.dst != 224.0.0.251 &&
  v Ethernet: Destination: 09:15:06 (00:0e:65:09:15:06), Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  > Destination: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  > Source: Apple_09:15:06 (00:e6:50:09:15:06)
  > Type: IPv4 (0x8000)
  v Internet Protocol Version 4, Src: 172.26.69.211, Dst: 193.136.9.254
  0100 .... + Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x920c (37388)
  v Flags: 0x2099 More fragments
    0... .... .... = Reserved bit: Not set
    .0. .... .... = Don't fragment: Not set
    ..1.... .... = More fragments: Set
    ...0 0000 1011 1001 = Fragment offset: 185
  v Time to live: 1
  > [Expert Info (Note/Sequence): "Time To Live" only 1]
  Protocol: ICMP (1)
  Header checksum: 0x43e8 [validation disabled]
  [Header checksum status: Unverified]
  Source: 172.26.69.211
  Destination: 193.136.9.254
  Reassembled IPv4 in frame: 3
  > Data (1480 bytes)

```

Figura 27.

Resposta: Podemos confirmar que não se trata do 1º fragmento pois o valor de offset é diferente de 0 e apresenta-se, neste caso, como 185(octetos). Há ainda mais fragmentos pois conseguimos ver na flag que “More Fragments : Set” (assume valor 1).

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

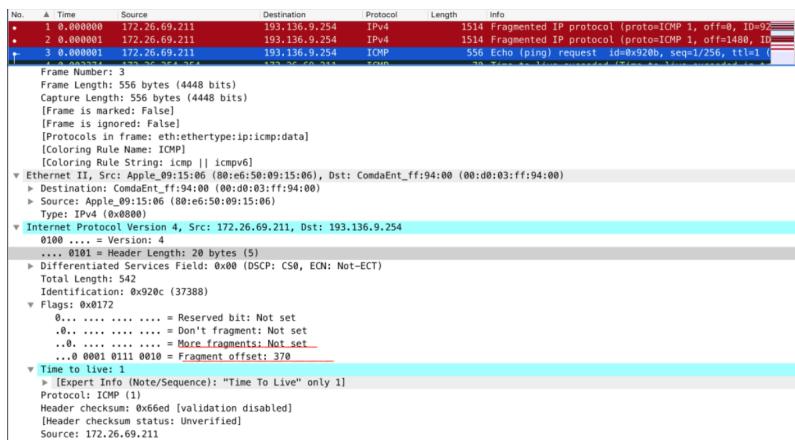


Figura 28.

Resposta: Foram criados 3 fragmentos a partir do datagrama original. Como estes 3 fragmentos vão ser reconstruídos na trama 3, a trama 3 corresponderá ao último fragmento do datagrama original. Através da flag “More Fragments: Not Set” que assume valor 0 concluímos que não há mais fragmentos. Através do offset diferente de a 0 verifica-se que não se pode tratar do primeiro fragmento logo o fragmento em questão só pode ser o último.

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.

Resposta: Os campos que mudam no cabeçalho IP entre os diferentes fragmentos do datagrama original são o offset dos fragmentos e a identificação de cada um destes. Através do offset, a reconstrução é possível assim que obtemos o último fragmento, uma vez que o offset fornece a ordem e posição de cada fragmento de modo a obtermos o datagrama original que queríamos transmitir em rede.

3 Parte 2

Considere que a organização MIEI-RC é constituída por três departamentos (A, B, e C) e cada departamento possui um router de acesso à sua rede local. Estes routers de acesso (Ra, Rb, e Rc) estão interligados entre si por ligações Ethernet a 1Gbps, formando um anel. Por sua vez, existe um servidor (S1) na rede do departamento C e, pelo menos, três laptops por departamento, interligados ao router respetivo através de um comutador (switch). S1 tem uma ligação a 1Gbps e os laptops ligações a 100Mbps. Considere apenas a existência de um comutador por departamento. A conectividade IP externa da organização é assegurada através de um router de acesso Rext conectado a Rc por uma ligação ponto-a-ponto a 10 Gbps.

3.1 Exercicio 1

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.

Resposta:

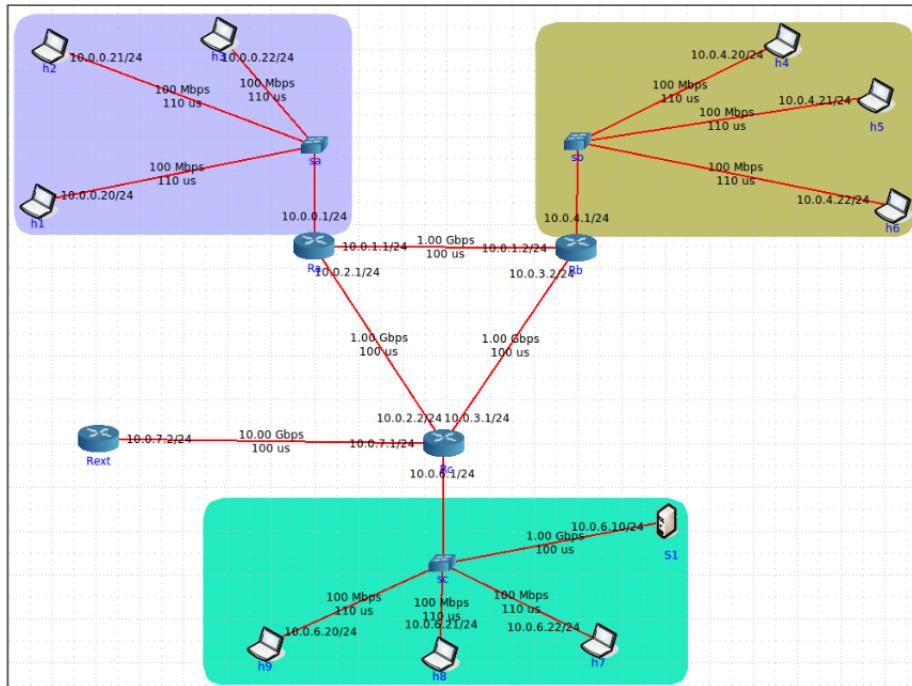


Figura 29. Exemplo de uma rede montada no CORE.

b) Tratam-se de endereços públicos ou privados? Porquê?

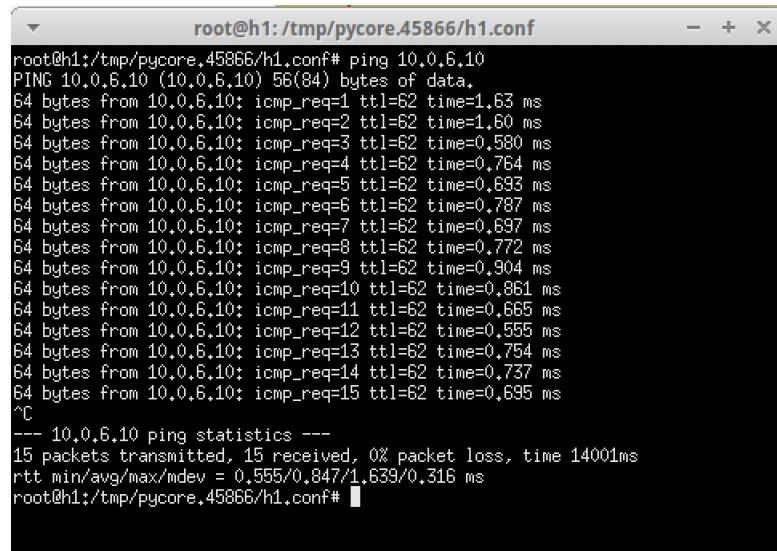
Resposta: Os endereços atribuídos pelo CORE são **privados**, porque caso fossem públicos podiam causar conflito. Como estes endereços, publicamente, podem ter os seus donos, caso estes endereços fossem públicos a interferência era grande.

c) Porque razão não é atribuído um endereço IP aos switches?

Resposta: Os switches, neste caso, funcionam como um repetidor, ou seja, repetem e alargam o sinal da rede que lhes é fornecida, portanto não precisam de adquirir um endereço de IP pois a rede que eles propagam já tem um endereço. Em suma, os switches trabalham na camada do nível 2, camada essa que não tem endereços IP.

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

Resposta: Como é possível verificar nas figuras a seguir, ao executar o comando **ping**, vemos que existe conexão entre um laptop de cada departamento e o servidor do departamento C.



```
root@h1:/tmp/pycore.45866/h1.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_req=1 ttl=62 time=1.63 ms
64 bytes from 10.0.6.10: icmp_req=2 ttl=62 time=1.60 ms
64 bytes from 10.0.6.10: icmp_req=3 ttl=62 time=0.580 ms
64 bytes from 10.0.6.10: icmp_req=4 ttl=62 time=0.764 ms
64 bytes from 10.0.6.10: icmp_req=5 ttl=62 time=0.693 ms
64 bytes from 10.0.6.10: icmp_req=6 ttl=62 time=0.787 ms
64 bytes from 10.0.6.10: icmp_req=7 ttl=62 time=0.697 ms
64 bytes from 10.0.6.10: icmp_req=8 ttl=62 time=0.772 ms
64 bytes from 10.0.6.10: icmp_req=9 ttl=62 time=0.904 ms
64 bytes from 10.0.6.10: icmp_req=10 ttl=62 time=0.881 ms
64 bytes from 10.0.6.10: icmp_req=11 ttl=62 time=0.665 ms
64 bytes from 10.0.6.10: icmp_req=12 ttl=62 time=0.555 ms
64 bytes from 10.0.6.10: icmp_req=13 ttl=62 time=0.754 ms
64 bytes from 10.0.6.10: icmp_req=14 ttl=62 time=0.737 ms
64 bytes from 10.0.6.10: icmp_req=15 ttl=62 time=0.695 ms
^C
--- 10.0.6.10 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 14001ms
rtt min/avg/max/mdev = 0.555/0.847/1.639/0.316 ms
root@h1:/tmp/pycore.45866/h1.conf#
```

Figura 30. Conetividade entre o laptop h1 com o servidor do departamento C.

```

root@h4:/tmp/pycore.45866/h4.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_req=1 ttl=62 time=1.33 ms
64 bytes from 10.0.6.10: icmp_req=2 ttl=62 time=0.725 ms
64 bytes from 10.0.6.10: icmp_req=3 ttl=62 time=0.841 ms
64 bytes from 10.0.6.10: icmp_req=4 ttl=62 time=0.912 ms
64 bytes from 10.0.6.10: icmp_req=5 ttl=62 time=1.18 ms
64 bytes from 10.0.6.10: icmp_req=6 ttl=62 time=0.885 ms
64 bytes from 10.0.6.10: icmp_req=7 ttl=62 time=0.934 ms
64 bytes from 10.0.6.10: icmp_req=8 ttl=62 time=2.46 ms
64 bytes from 10.0.6.10: icmp_req=9 ttl=62 time=0.590 ms
64 bytes from 10.0.6.10: icmp_req=10 ttl=62 time=0.595 ms
64 bytes from 10.0.6.10: icmp_req=11 ttl=62 time=2.83 ms
64 bytes from 10.0.6.10: icmp_req=12 ttl=62 time=0.700 ms
64 bytes from 10.0.6.10: icmp_req=13 ttl=62 time=0.778 ms
^C
--- 10.0.6.10 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12008ms
rtt min/avg/max/mdev = 0.590/1.136/2.836/0.681 ms
root@h4:/tmp/pycore.45866/h4.conf#

```

Figura 31. Conetividade entre o laptop h4 com o servidor do departamento C.

```

root@h7:/tmp/pycore.45866/h7.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_req=1 ttl=64 time=0.802 ms
64 bytes from 10.0.6.10: icmp_req=2 ttl=64 time=0.833 ms
64 bytes from 10.0.6.10: icmp_req=3 ttl=64 time=0.549 ms
64 bytes from 10.0.6.10: icmp_req=4 ttl=64 time=0.470 ms
64 bytes from 10.0.6.10: icmp_req=5 ttl=64 time=0.439 ms
64 bytes from 10.0.6.10: icmp_req=6 ttl=64 time=0.444 ms
64 bytes from 10.0.6.10: icmp_req=7 ttl=64 time=0.331 ms
64 bytes from 10.0.6.10: icmp_req=8 ttl=64 time=0.467 ms
64 bytes from 10.0.6.10: icmp_req=9 ttl=64 time=0.388 ms
64 bytes from 10.0.6.10: icmp_req=10 ttl=64 time=0.353 ms
64 bytes from 10.0.6.10: icmp_req=11 ttl=64 time=0.461 ms
64 bytes from 10.0.6.10: icmp_req=12 ttl=64 time=0.388 ms
64 bytes from 10.0.6.10: icmp_req=13 ttl=64 time=0.431 ms
64 bytes from 10.0.6.10: icmp_req=14 ttl=64 time=0.382 ms
^C
--- 10.0.6.10 ping statistics ---
14 packets transmitted, 14 received, 0% packet loss, time 13000ms
rtt min/avg/max/mdev = 0.331/0.481/0.833/0.148 ms
root@h7:/tmp/pycore.45866/h7.conf#

```

Figura 32. Conetividade entre o laptop h7 com o servidor do departamento C.

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

Resposta: Na imagem a seguir, voltamos a executar o comando ping, para verificar se existia conectividade entre o router de acesso, e o servidor do departamento C. Depois de executado o comando podemos ver que existe essa conectividade.

```

root@Rext:/tmp/pycore.46010/Rext.conf
root@Rext:/tmp/pycore.46010/Rext.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_req=1 ttl=63 time=0.620 ms
64 bytes from 10.0.6.10: icmp_req=2 ttl=63 time=0.475 ms
64 bytes from 10.0.6.10: icmp_req=3 ttl=63 time=0.554 ms
64 bytes from 10.0.6.10: icmp_req=4 ttl=63 time=0.484 ms
64 bytes from 10.0.6.10: icmp_req=5 ttl=63 time=0.486 ms
64 bytes from 10.0.6.10: icmp_req=6 ttl=63 time=0.465 ms
64 bytes from 10.0.6.10: icmp_req=7 ttl=63 time=0.548 ms
64 bytes from 10.0.6.10: icmp_req=8 ttl=63 time=0.552 ms
64 bytes from 10.0.6.10: icmp_req=9 ttl=63 time=0.442 ms
64 bytes from 10.0.6.10: icmp_req=10 ttl=63 time=0.476 ms
64 bytes from 10.0.6.10: icmp_req=11 ttl=63 time=0.447 ms
^C
--- 10.0.6.10 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 9996ms
rtt min/avg/max/mdev = 0.442/0.504/0.620/0.057 ms
root@Rext:/tmp/pycore.46010/Rext.conf#

```

Figura 33. Conetividade entre o Rext e com o servidor do departamento C.

3.2 Exercicio 2

Para o router e um laptop do departamento A:

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

Resposta:

Kernel IP routing table						
Destination	Gateway	Genmask	Flags	MSS	Window	irtt Iface
0.0.0.0	10.0.2.2	0.0.0.0	UG	0	0	0 eth0
10.0.2.0	0.0.0.0	255.255.255.0	U	0	0	0 eth0
169.254.0.0	0.0.0.0	255.255.0.0	U	0	0	0 eth0

Figura 34. Tabela de encaminhamento de um portátil

Kernel IP routing table						
Destination	Gateway	Genmask	Flags	MSS	Window	irtt Iface
0.0.0.0	10.0.2.2	0.0.0.0	UG	0	0	0 eth0
10.0.2.0	0.0.0.0	255.255.255.0	U	0	0	0 eth0
169.254.0.0	0.0.0.0	255.255.0.0	U	0	0	0 eth0

Figura 35. Tabela de encaminhamento do router

A primeira coluna especifica o destino da conexão. A segunda coluna mostra a 'gateway' usada pela conexão. A coluna 'Genmask' mostra a máscara de rede. A coluna 'Flags' mostra as flags da conexão ('U'- route is up; 'G'- use gateway). A coluna 'MSS' representa o Maximum Segment Size desta rota. 'Window' especifica o tamanho da janela para conexões TCP nesta rota. A coluna 'irtt' mostra o Initial Round Trip Time. A última coluna diz-nos qual a interface que está a ser usada no encaminhamento desta conexão.

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

Resposta: O protocolo OSPF está ativo no router mas não nos hosts, logo é usado encaminhamento dinâmico para os routers e encaminhamento estático para os hosts.

```
root@n6:/tmp/pycore.52750/n6.conf# ps -ef
UID      PID  PPID  C STIME TTY          TIME CMD
root      1      0  0 09:38 ?        00:00:00 /usr/sbin/vnoded -v -c /tmp/pyco
root     16      1  0 09:38 pts/6   00:00:00 /bin/bash
root     26     16  0 09:39 pts/6   00:00:00 ps -ef
root@n6:/tmp/pycore.52750/n6.conf#
```

Figura 36. Output do comando 'ps -ef' num host da rede

```
root@Ra:/tmp/pycore.52750/Ra.conf# ps -ef
UID      PID  PPID  C STIME TTY          TIME CMD
root      1      0  0 09:38 ?        00:00:00 /usr/sbin/vnoded -v -c /tmp/pyco
root     60      1  0 09:38 ?        00:00:00 /usr/lib/quagga/zebra -u root -g
root     68      1  0 09:38 ?        00:00:00 /usr/lib/quagga/ospfd -u root -g
root     69      1  0 09:38 ?        00:00:00 /usr/lib/quagga/ospf6d -u root -
root     75      1  0 09:39 pts/8   00:00:00 /bin/bash
root     85     75  0 09:39 pts/8   00:00:00 ps -ef
root@Ra:/tmp/pycore.52750/Ra.conf#
```

Figura 37. Output do comando 'ps -ef' num router da rede

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 C. Use o comando route delete para o efeito. Que implicações tem esta medida para os utilizadores da empresa que acedem ao servidor. Justifique.

Resposta: Os hosts que se encontram na mesma subrede que S1 não serão afetados, no entanto os utilizadores que se encontrem noutra rede que não a do departamento C não terão acesso a S1, visto que deixou de existir a única rota de acesso ao servidor.

```
root@n9:/tmp/pycore.55058/n9.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_req=1 ttl=64 time=1.38 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=64 time=0.290 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=64 time=0.281 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=64 time=0.281 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=64 time=0.320 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 0.281/0.510/1.380/0.435 ms
```

Figura 38. Comando 'ping' de um host da mesma subrede de S1 (há conexão)

```
root@n5:/tmp/pycore.52751/n5.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 ---
29 packets transmitted, 0 received, 100% packet loss, time 28078ms
root@n5:/tmp/pycore.52751/n5.conf# █
```

Figura 39. Comando 'ping' de um host no departamento A para S1 (sem resposta)

- 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 registe os comandos que usou.

Resposta:

```
root@S1:/tmp/pycore.55130/S1.conf# route add default gw 10.0.5.1
root@S1:/tmp/pycore.55130/S1.conf#
root@S1:/tmp/pycore.55130/S1.conf#
root@S1:/tmp/pycore.55130/S1.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@S1:/tmp/pycore.55130/S1.conf# █
```

Figura 40. Adição de uma nova rota; nova tabela de encaminhamento

- 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.

Resposta:

```
root@n5:/tmp/pycore.55130/n5.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_req=1 ttl=62 time=0.914 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=62 time=0.330 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=62 time=0.306 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=62 time=0.298 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=62 time=0.304 ms
^C
--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3998ms
rtt min/avg/max/mdev = 0.298/0.430/0.914/0.242 ms
root@n5:/tmp/pycore.55130/n5.conf#
```

Figura 41. Comando 'ping', executado a partir de um host fora da subrede de S1. A nova tabela de encaminhamento está visível na **Figura 40**

3.3 Exercicio 3

Definição de Sub-redes Por forma a minimizar a falta de endereços IPv4 é comum a utilização de sub-redes. Além disso, a definição de sub-redes permite uma melhor organização do espaço de endereçamento das redes em questão. Para definir endereços de sub-rede é necessário usar a parte prevista para endereçamento de host, não sendo possível alterar o endereço de rede original. Recorda-se que o subnetting, ao recorrer ao espaço de endereçamento para host, implica que possam ser endereçados menos hosts. Considere a topologia definida anteriormente. Assuma que o endereçamento entre os routers se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.

a) Considere que dispõe apenas do endereço de rede IP 172.12.48.0/20, em que 12 é o decimal correspondendo ao seu número de grupo (PL12). 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.

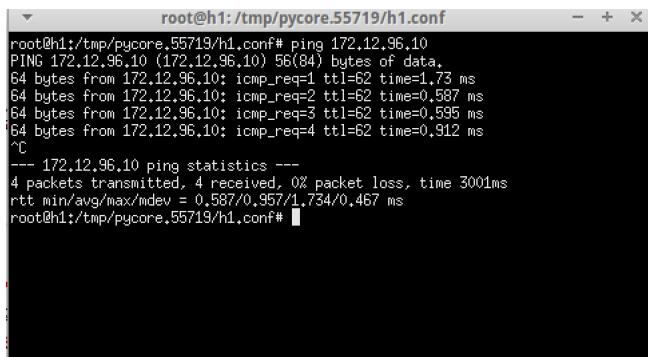
Resposta: Para definir o novo esquema de sub-redes, decidimos usar um esquema de 4 bits para indentificar as sub-redes no endereço IP ficando assim com uma mascara de subrede /24. Desta forma, conseguimos endereços suficientes para identificar todos os departamentos que constituem esta rede e futuros departamentos que possam vir a ser construídos, assim asseguramos a capacidade de expansão do mapa de rede. Tivemos de definir as nossas sub-redes de 16 em 16, ou seja, a primeira sub-rede corresponde ao departamento A e tem como endereço 172.12.64.1, a segunda sub-rede corresponde ao departamento B e tem como endereço 172.12.80.1, a terceira corresponde ao departamento C com endereço 172.12.96.1.

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

Resposta: A mascara de subrede que usamos é 24, que apresenta como endereço 255.255.255.0. O número total de hosts que se podem ligar a estas sub-redes são $(2^8) - 2$ (devido ao endereço que identifica todas as redes e ao endereço de broadcast) = 254.

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

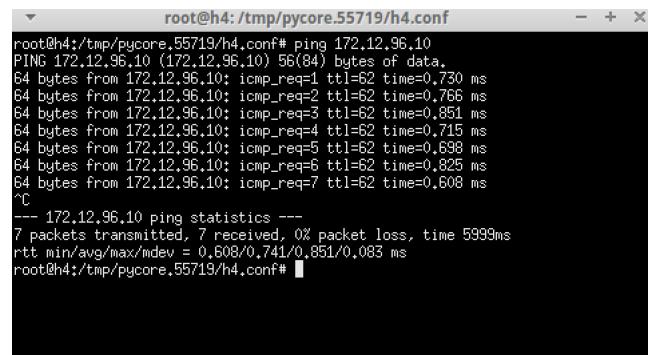
Resposta:



```
root@h1:/tmp/pycore.55719/h1.conf
root@h1:/tmp/pycore.55719/h1.conf# ping 172.12.96.10
PING 172.12.96.10 (172.12.96.10) 56(84) bytes of data,
64 bytes from 172.12.96.10: icmp_req=1 ttl=62 time=0.73 ms
64 bytes from 172.12.96.10: icmp_req=2 ttl=62 time=0.587 ms
64 bytes from 172.12.96.10: icmp_req=3 ttl=62 time=0.595 ms
64 bytes from 172.12.96.10: icmp_req=4 ttl=62 time=0.912 ms
^C
--- 172.12.96.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 0.587/0.957/1.734/0.467 ms
root@h1:/tmp/pycore.55719/h1.conf#
```

Figura 42. Execução do comando ping do laptop h1 para o servidor 1, e consequente conexão

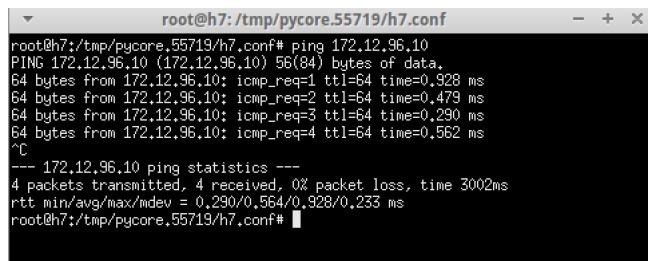
Resposta:



```
root@h4:/tmp/pycore.55719/h4.conf
root@h4:/tmp/pycore.55719/h4.conf# ping 172.12.96.10
PING 172.12.96.10 (172.12.96.10) 56(84) bytes of data,
64 bytes from 172.12.96.10: icmp_req=1 ttl=62 time=0.730 ms
64 bytes from 172.12.96.10: icmp_req=2 ttl=62 time=0.766 ms
64 bytes from 172.12.96.10: icmp_req=3 ttl=62 time=0.851 ms
64 bytes from 172.12.96.10: icmp_req=4 ttl=62 time=0.715 ms
64 bytes from 172.12.96.10: icmp_req=5 ttl=62 time=0.898 ms
64 bytes from 172.12.96.10: icmp_req=6 ttl=62 time=0.825 ms
64 bytes from 172.12.96.10: icmp_req=7 ttl=62 time=0.608 ms
^C
--- 172.12.96.10 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 5999ms
rtt min/avg/max/mdev = 0.608/0.741/0.851/0.083 ms
root@h4:/tmp/pycore.55719/h4.conf#
```

Figura 43. Execução do comando ping do laptop h4 para o servidor 1, e consequente conexão

Resposta:



```
root@h7:/tmp/pycore.55719/h7.conf
root@h7:/tmp/pycore.55719/h7.conf# ping 172.12.96.10
PING 172.12.96.10 (172.12.96.10) 56(84) bytes of data,
64 bytes from 172.12.96.10: icmp_req=1 ttl=64 time=0.928 ms
64 bytes from 172.12.96.10: icmp_req=2 ttl=64 time=0.479 ms
64 bytes from 172.12.96.10: icmp_req=3 ttl=64 time=0.290 ms
64 bytes from 172.12.96.10: icmp_req=4 ttl=64 time=0.562 ms
^C
--- 172.12.96.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.290/0.564/0.928/0.233 ms
root@h7:/tmp/pycore.55719/h7.conf#
```

Figura 44. Execução do comando ping do laptop h7 para o servidor 1, e consequente conexão

4 Conclusions

Ao realizar deste trabalho pratico no ambito da Unidade Curricular Redes de Computadores, deparamo-nos com os conceitos mencionados nas aulas práticas da disciplina e aprendemos a interagir, interpretar e manipular os conceitos fundamentais na estrutura inicial de redes. Nomeadamente deparamo-nos com conceitos relativos a datagramas, fragmentação de mensagens, encaminhamento , endereçamento IP e também subnetting, bem como os conceitos intrínsecos que sao fundamentais à contrução das bases essenciais para realização de atividades relacionadas com a disciplina.

No desenvolvimento deste trabalho, conseguimos, perante uma captura, analisar o tráfego ICMP, assim como o tempo de vida de determinados pacotes até a sua chegada á interface para onde o enviamos, tal como os recebidos. O uso do simulador da rede CORE e do analisador de tráfego Wireshark permitiu-nos abordar a comunicação entre várias interfaces e diversas configurações de redes construídas com a utilização de router, switch, servidores e hosts.