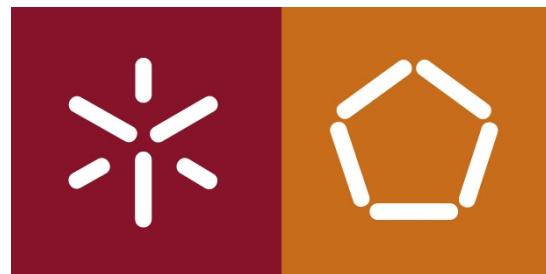


UNIVERSIDADE DO MINHO

Mestrado Integrado em Engenharia Informática



---

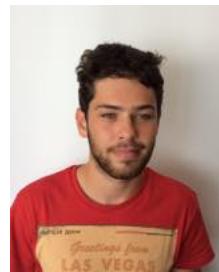
## REDES DE COMPUTADORES

TP2 – Protocolo IPv4

---



Davide Matos (A80970)



Francisco Freitas (A81580)

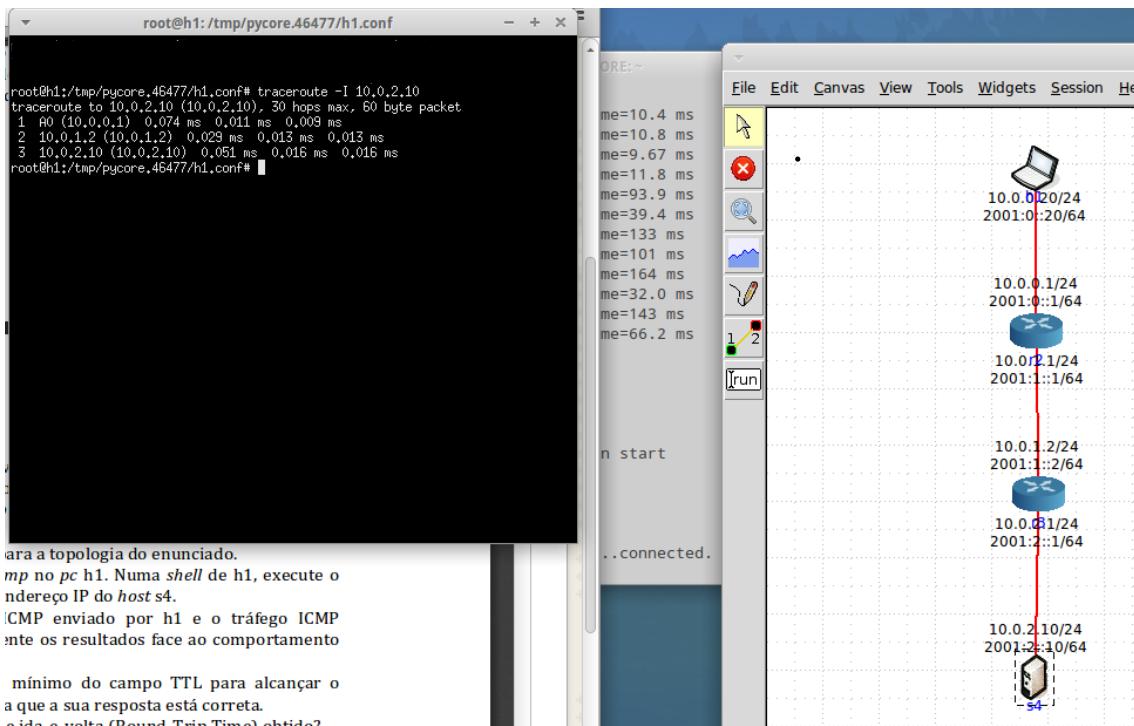


Maria Dias (A81611)

## 1ª PARTE

1. 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.

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

Executando o comando “traceroute -l 10.0.2.10” a partir do host h1, são devolvidos 3 tempos relativos aos 3 pacotes enviados para o host s4. Notamos que o tempo do primeiro pacote é sempre superior aos restantes.

Inicialmente, são enviados 3 pacotes do endereço 10.0.0.20 (h1) com destino a 10.0.2.10 (s4), com TTL = 1. Conseguimos observar isto a partir da linha 32 da captura de ecrã que apresentamos a seguir. O TTL é o número de saltos entre dispositivos (como routers) que um pacote pode dar antes de chegar ao seu destino. A cada “salto” que o pacote faz, o TTL é decrementado uma unidade. Quando este valor chega a zero, o pacote é descartado.

No entanto, estes não passaram do router com o endereço 10.0.0.1, que retornou uma mensagem com o conteúdo “Time to live exceeded in-transit”. Esta mensagem significa que o pacote foi descartado devido ao campo TTL ter atingido o valor nulo.

De seguida, são novamente enviados 3 pacotes, desta feita com TTL = 2, sendo os mesmos descartados no router 10.0.1.2, que envia as mensagens de erro.

Finalmente, são lançados 3 pacotes com TTL = 3, que chegam ao destino final com TTL = 1, por isso não vemos mais nenhuma mensagem de erro.

No.	Time	Source	Destination	Protocol	Length	Info
30	99.17.178.100:00:00:aa:00:04	7E80:200:ff:feaa:5ff0:2:5	192.168.1.10:00:00:aa:00:04	OSPF	96	OSPF Hello Packet
31	99.554892 00:00:00:aa:00:05	7E80:200:ff:feaa:5ff0:2:5	Broadcast	ARP	42	42 who has 10.0.0.17 Tell 10.0.0.20
31	99.554892 00:00:00:aa:00:05	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ARP	42	10.0.0.1 is at 00:00:00:aa:00:05
32	99.554892 00:00:00:aa:00:04	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=1/256, ttl=1
33	99.554953 10.0.1.1	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
34	99.554964 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=2/512, ttl=1
34	99.554964 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
36	99.554976 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=3/768, ttl=1
36	99.554976 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
38	99.554987 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=4/1024, ttl=1
38	99.555013 10.0.1.2	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
40	99.555019 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=5/1280, ttl=1
41	99.555029 10.0.1.2	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
42	99.555033 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=6/1536, ttl=1
42	99.555064 10.0.1.2	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
44	99.555051 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=7/1792, ttl=1
45	99.555051 10.0.1.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	74	Echo (ping) reply id=0x0049, seq=7/1792, ttl=62
46	99.555107 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=8/2048, ttl=1
47	99.555120 10.0.2.10	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	74	Echo (ping) reply id=0x0049, seq=8/2048, ttl=62
48	99.555125 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=9/2304, ttl=1
49	99.555138 10.0.2.10	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	74	Echo (ping) reply id=0x0049, seq=9/2304, ttl=62
50	99.555143 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=10/2560, ttl=1
51	99.555143 10.0.2.10	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	74	Echo (ping) reply id=0x0049, seq=10/2560, ttl=62
52	99.555161 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=11/2816, ttl=1
53	99.555172 10.0.2.10	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	74	Echo (ping) reply id=0x0049, seq=11/2816, ttl=62
54	99.555176 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=12/3072, ttl=1
55	99.555189 10.0.2.10	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	74	Echo (ping) reply id=0x0049, seq=12/3072, ttl=62
56	99.555191 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=13/3232, ttl=1
57	99.555206 10.0.2.10	7E80:200:ff:feaa:5ff0:2:5	10.0.2.20	ICMP	74	Echo (ping) reply id=0x0049, seq=13/3232, ttl=62
58	99.555210 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) request id=0x0049, seq=14/3584, ttl=1
59	99.555210 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Echo (ping) reply id=0x0049, seq=14/3584, ttl=62
60	99.555237 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Time-to-live exceeded (Time to live exceeded in transit)
60	99.555237 10.0.2.20	7E80:200:ff:feaa:5ff0:2:5	10.0.2.10	ICMP	74	Time-to-live exceeded (Time to live exceeded in transit)
61	101.151484 fe80:200:ff:feaa:5ff0:2:5	192.168.1.10:00:00:aa:00:04	OSPF	90	Hello Packet	

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.

O valor inicial mínimo do campo TTL deve ser 3, para poder alcançar o destino s4, porque a partir do TTL 3 não se verifica a receção de mensagens de erro, mas sim a receção de replies por parte do destino. Podemos constatar isto na captura de ecrã apresentada na alínea anterior.

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

Observando a primeira captura de ecrã, conseguimos retirar da linha 3 os últimos 3 tempos obtidos (que correspondem aos valores de round-trip time de cada um dos 3 pacotes desde a origem até ao destino) e verificamos que o valor médio do tempo de ida e volta é de  $((0.051 + 0.016 + 0.016)/3 = 0.028\text{ms}$ .

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

O endereço IP da interface ativa do computador usado é 10.0.2.15 e é indicado pelo campo “Source” destacado na imagem seguinte.

228	538.876597.193.136.16.65	10.0.2.15	DNS	247 Standard query response A 193.136.9.240
229	538.877264.10.0.2.15	193.136.9.240	ICMP	74 Echo (ping) request id=0x087e, seq=2/12, ttl=1
230	538.877264.10.0.2.15	193.136.9.240	ICMP	74 Echo (ping) request id=0x087e, seq=2/12, ttl=1
231	538.877359.10.0.2.15	10.0.2.15	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
232	538.877424.10.0.2.15	193.136.9.240	ICMP	74 Echo (ping) request id=0x087e, seq=3/168, ttl=1
233	538.877515.10.0.2.15	10.0.2.15	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
234	538.877575.10.0.2.15	193.136.9.240	ICMP	74 Echo (ping) request id=0x087e, seq=4/1024, ttl=2
235	538.877643.10.0.2.15	10.0.2.15	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
236	538.877723.10.0.2.15	193.136.9.240	ICMP	74 Echo (ping) request id=0x087e, seq=5/1280, ttl=2
237	538.877828.10.0.2.15	193.136.9.240	ICMP	74 Echo (ping) request id=0x087e, seq=6/1536, ttl=2
> Frame 228: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)				
Ethernet II, Src: CadmusCo_70:5d:be (08:00:27:70:5d:be), Dst: RealtekAU_12:35:02 (52:54:00:12:35:02)				
Internet Protocol Version 4, Src: 10.0.2.15 (10.0.2.15), Dst: 193.136.9.240 (193.136.9.240)				
Version: 4				
Header length: 20 bytes				
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))				
Total Length: 60				
Identification: 0x148f (5263)				
Flags: 0x0				
Fragment offset: 0				
Time to live: 1				
Protocol: ICMP (1)				
Header checksum: 0xcbad [correct]				
Source: 193.136.9.240 (193.136.9.240)				
Destination: 193.136.9.240 (193.136.9.240)				
Internet Control Message Protocol				
0010	00 3c 14 8f 00 00 01 cd ab .....	c1 88 .....	..,.....,.....,	
0020	09 ff 08 09 79 fb 08 7e 00 01 48 49 4a 4b 4c 4d .....,.,.HJKLM	55 56 57 58 59 5a 5b 5c 5d .....		
0030	4e 4f 50 51 52 53 54 55 56 57 58 59 5a 5b 5c 5d NOPQRSTUVWXYZ[	5e .....		
0040	58 5f 60 61 62 63 64 65 66 67 .....	abccdf fg		
Source (ip_src),4 bytes				
Packets: 303 Displayed: 303 Marked: 0				
Profile: Default				

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

O valor do campo protocolo é 1, identificando o protocolo 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?

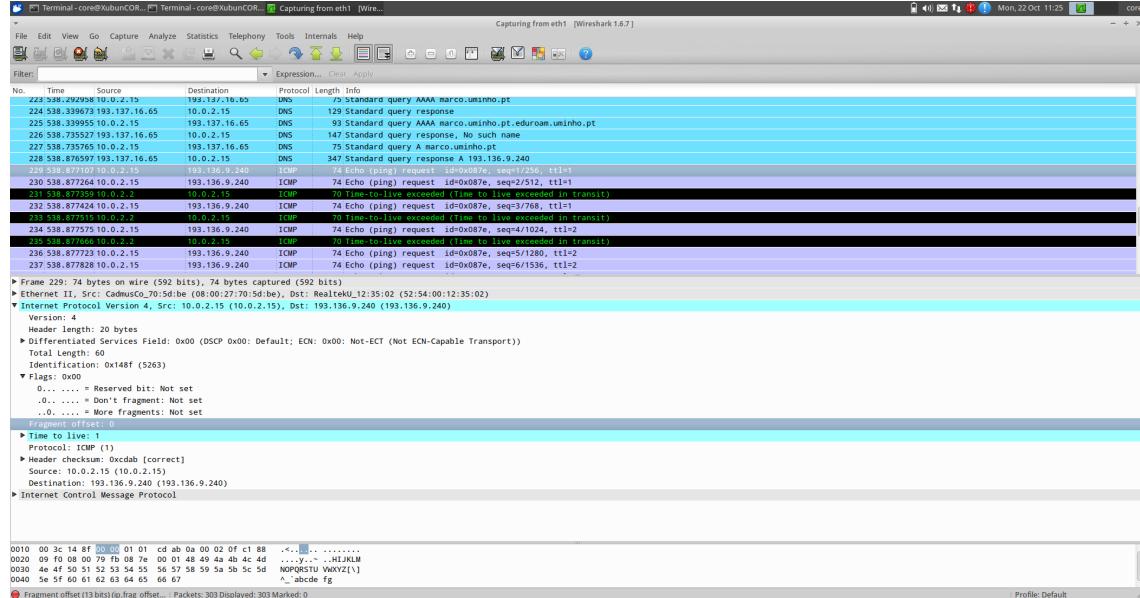
O cabeçalho IP(v4) tem 20 bytes. O campo “Total Length” indica o tamanho total do datagrama e tem valor 60. O cálculo do tamanho do payload faz-se subtraindo o tamanho do cabeçalho ao tamanho total do datagrama. Logo, o campo de dados do datagrama tem 40 bytes (= 60-20).

d. O datagrama IP foi fragmentado? Justifique.

O datagrama não foi fragmentado.

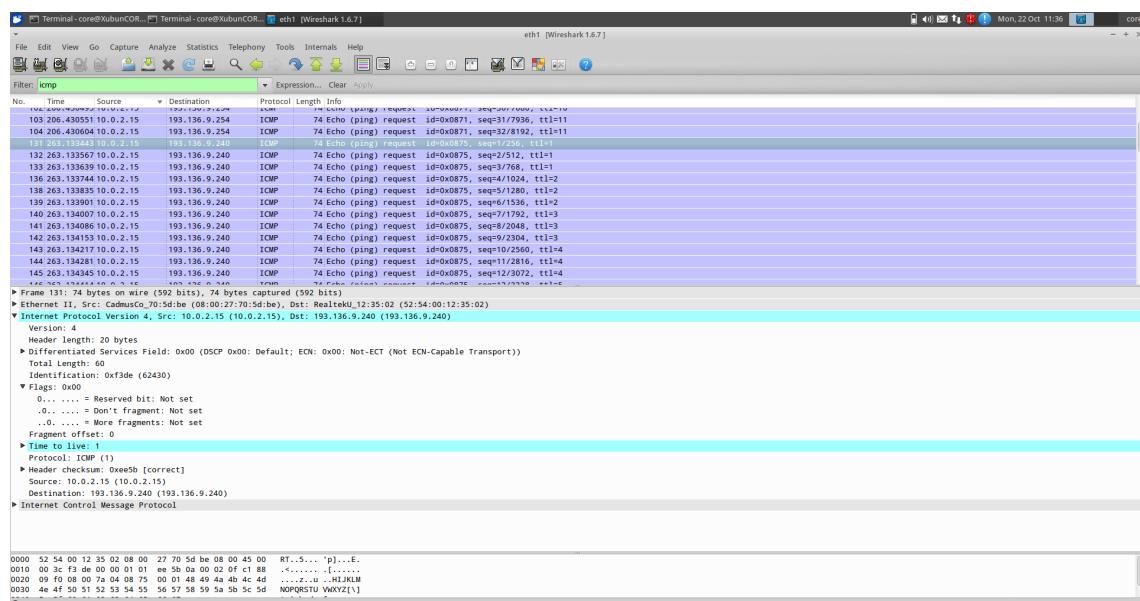
No indicador “Flags” encontra-se a flag “Fragment offset”, que indica a posição que o fragmento ocupa no datagrama original, e que neste caso tem o valor 0, o que significa que, existindo mais fragmentos, este é o primeiro de todos; e

também a flag “More fragments”, que tem valor 0, indicando que não existem mais fragmentos para além do próprio. Assim sendo, se este fragmento é o primeiro e se não existem mais fragmentos, concluímos que se trata do datagrama IP original, e que este não foi fragmentado.



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

Os campos que variam de pacote para pacote são o campo de identificação (ID) do datagrama e o TTL.

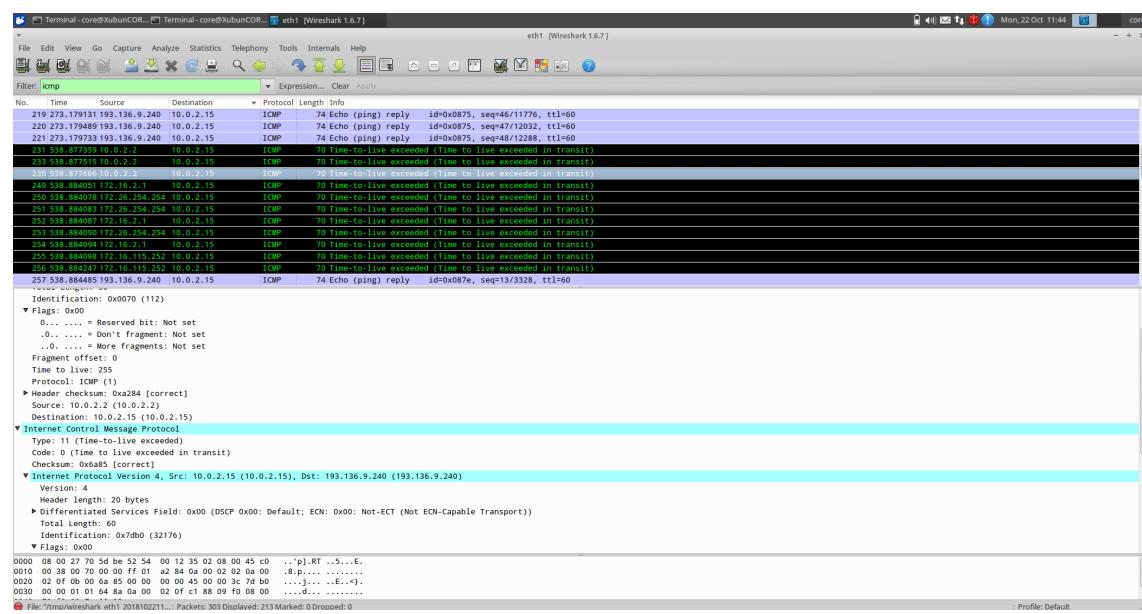


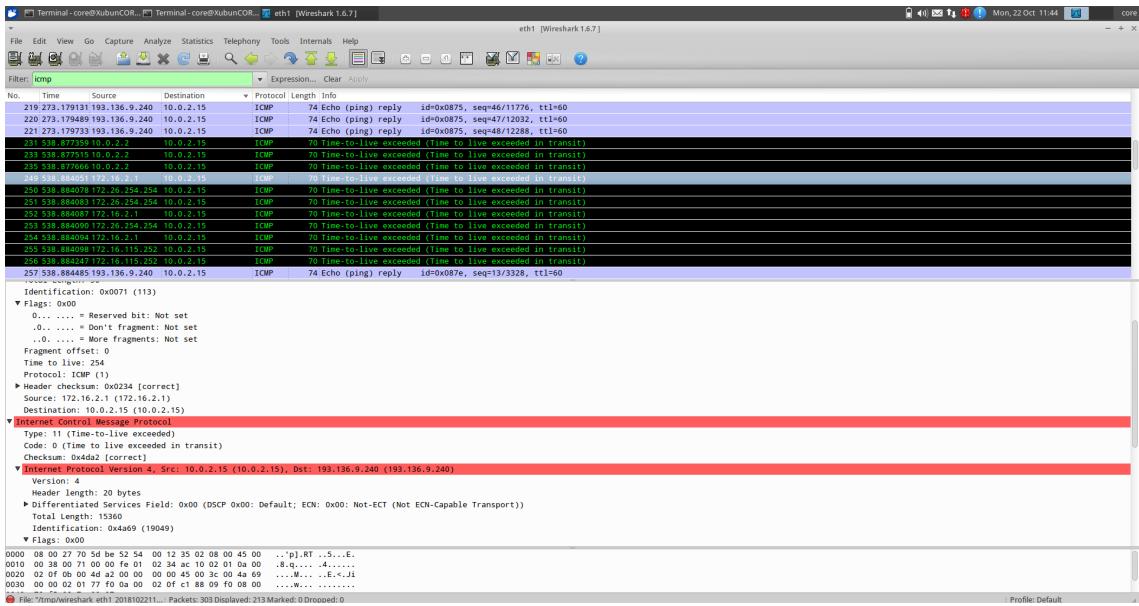
f. Observa algum padrão nos valores do campo de Identificação do datagrama IP e TTL?

Sim. Ambos são incrementados em 1 valor.

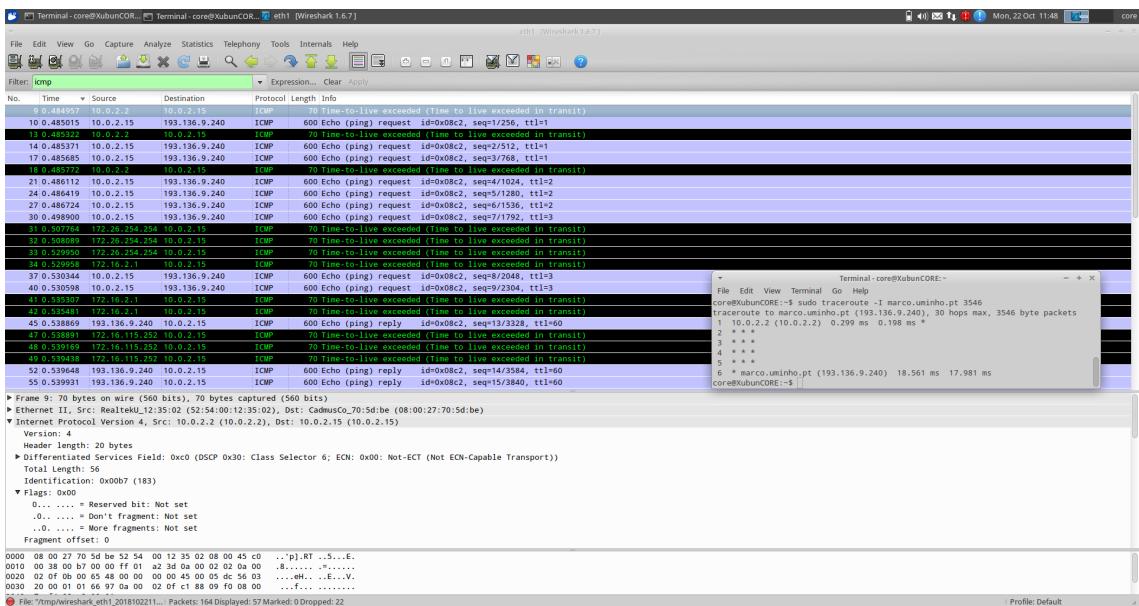
g. Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP TTL exceeded enviadas ao seu computador. Qual é o valor do campo TTL? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL exceeded enviados ao seu host? Porquê?

O valor do campo TTL é 255 e não permanece constante (por cada salto efetuado é-lhe decrementada uma unidade). Isto acontece porque, à medida que vão sendo emitidas as mensagens de erro (a partir dos routers mais distantes da rota que o pacote efetuou), aproximamo-nos da origem do nosso pacote, logo a “distância”, em forma de TTL, é cada vez menor neste caminho de “regresso” à fonte.



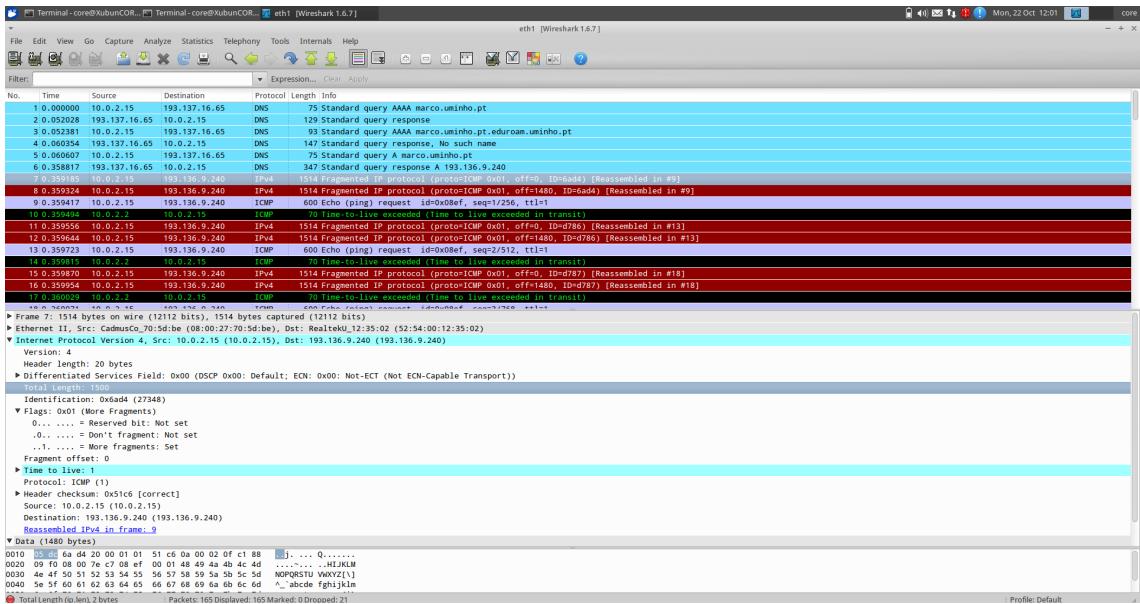


3. Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempo de captura. Observe o tráfego depois do tamanho de pacote ter sido definido para 3546 bytes.



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

O pacote inicial teve de ser fragmentado pois o MTU (Maximum Transport Unit) da rede é 1500 bytes, ou seja, o tamanho máximo do pacote permitido nesta rede é de 1500 bytes, e o tamanho do pacote que queremos enviar é de 3546 bytes. Para poder seguir nesta rede, o pacote teve de se dividir em 3 fragmentos mais pequenos.

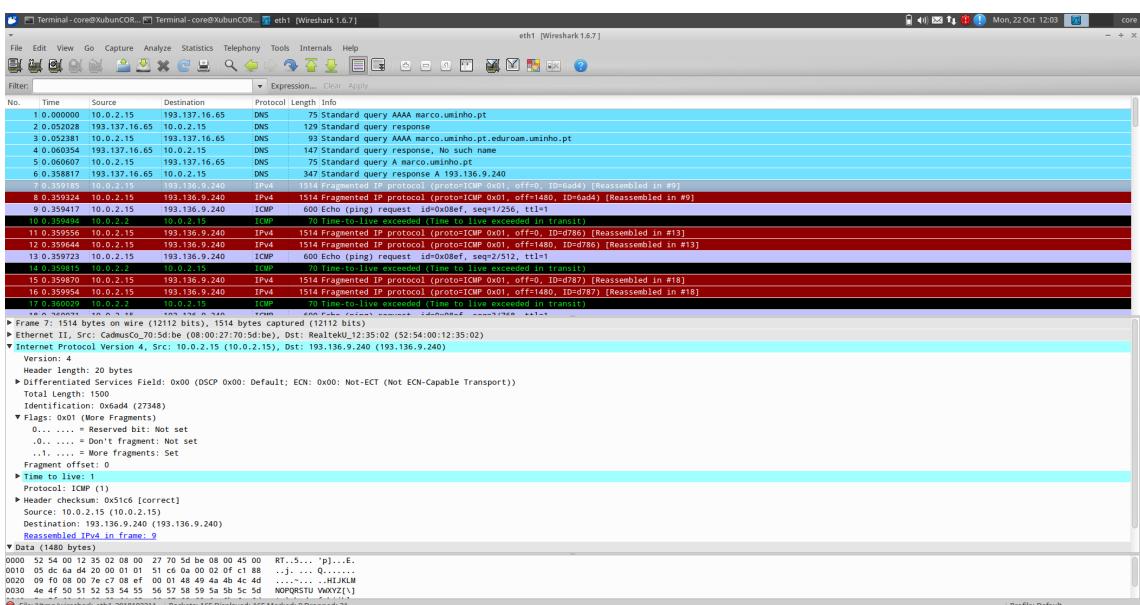


b. Imprima o primeiro fragmento do datagrama IP segmentado. Que informação no cabeçalho indica que o datagrama foi fragmentado? Que informação no cabeçalho IP indica que se trata do primeiro fragmento? Qual é o tamanho deste datagrama IP?

A flag “More Fragments” indica se existem mais fragmentos ou não. Neste caso, a flag tem o valor 1, pelo que existem mais fragmentos.

A flag “Fragment Offset” indica a posição que este fragmento irá ocupar quando o pacote original for reagrupado. Esta está a 0, indicando que este pacote é o primeiro.

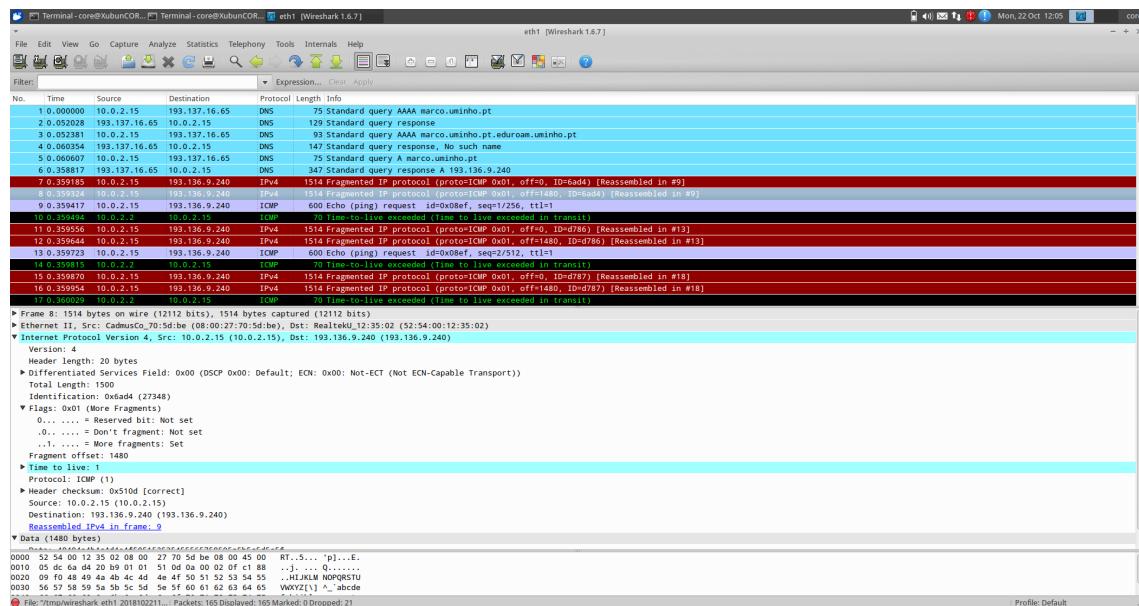
Este datagrama IP tem 1480 bytes, uma vez que o tamanho total do datagrama é 1500 bytes e o cabeçalho tem 20 bytes de comprimento ( $1500 - 20 = 1480$ ).



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?

A flag “Fragment Offset”, que tem o valor 1480, permite-nos afirmar que este não se trata do 1º fragmento. Vimos na alínea anterior que esta flag tinha o valor 0 e que o datagrama tinha 1480 bytes de comprimento. A flag do fragmento que estamos a analisar indica que este começa onde o anterior acaba ( $0+1480 = 1480$  bytes), concluindo assim que se trata do segundo fragmento.

De acordo com a flag “More fragments”, que tem o valor 1, existem mais fragmentos.

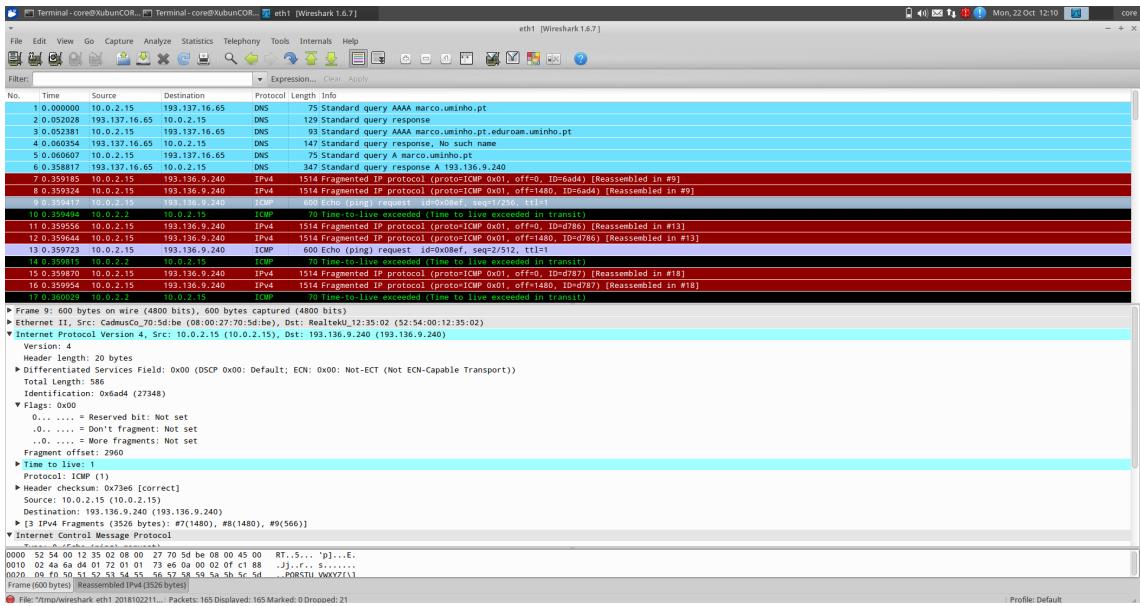


d. Quantos fragmentos foram criados a partir do datagrama original? Como se deteta o último fragmento correspondente ao datagrama original?

Foram criados 3 fragmentos a partir do datagrama original. O último fragmento tem a forma de mensagem ICMP. Todos os fragmentos correspondentes a um mesmo datagrama têm um número de identificação igual (neste caso, ...).

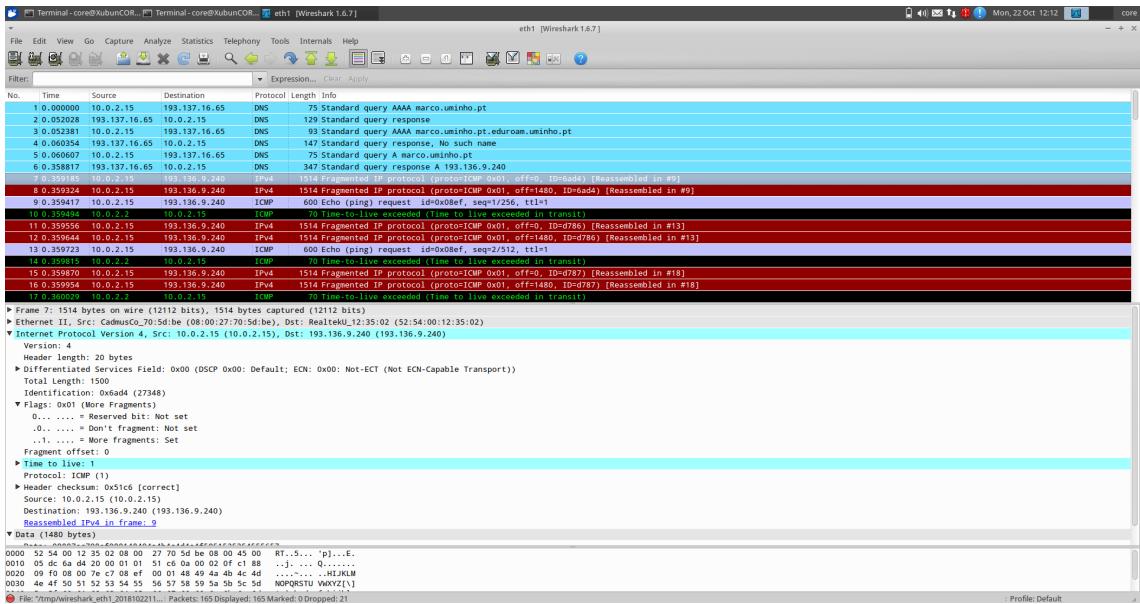
A flag “Fragment offset” deste fragmento é 2960 bytes, o que seria de esperar, pois a partir do offset do primeiro segmento, e somando os tamanhos dos fragmentos anteriores, obtemos ( $0+1480+1480 = 2960$  bytes).

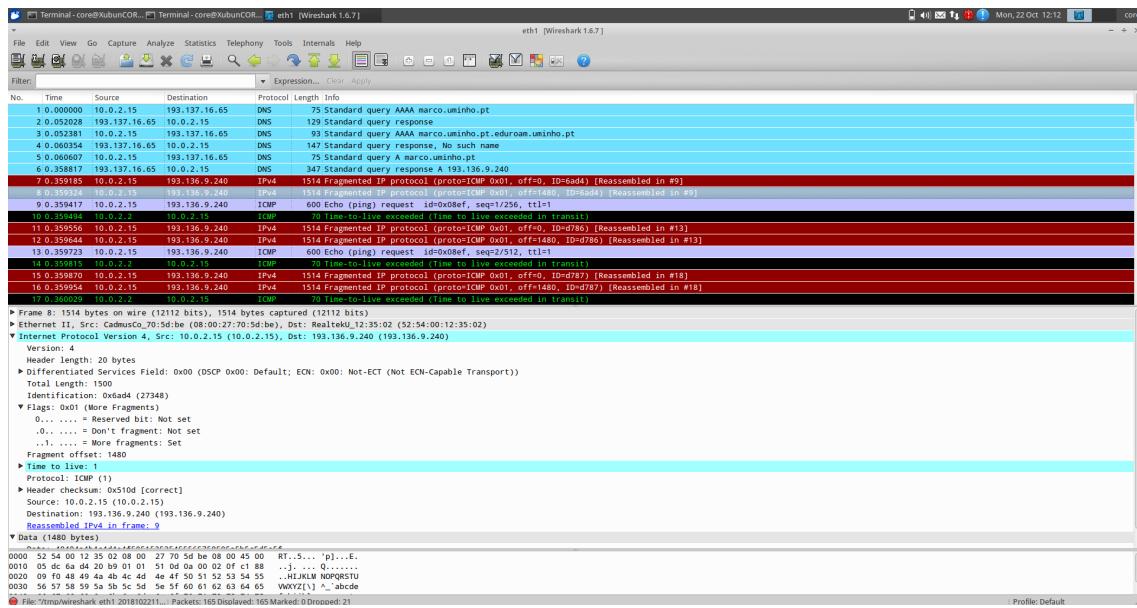
Por fim, a flag “More Fragments” é igual a 0, indicando que não existem mais fragmentos.



e. Indique, resumindo, os campos que mudam no cabeçalho IP entre os diferentes fragmentos, e explique a forma como essa informação permite reconstruir o datagrama original.

Os únicos campos que mudam entre os cabeçalhos IP dos diferentes fragmentos são as flags “Fragment offset” e “More fragments”. A primeira permite-nos identificar a posição do fragmento no datagrama original e a segunda indica se existem ou não mais fragmentos na rede. Desta forma, é possível reagrupar o pacote original, agrupando os fragmentos por ordem crescente do valor da flag “Fragment offset”.



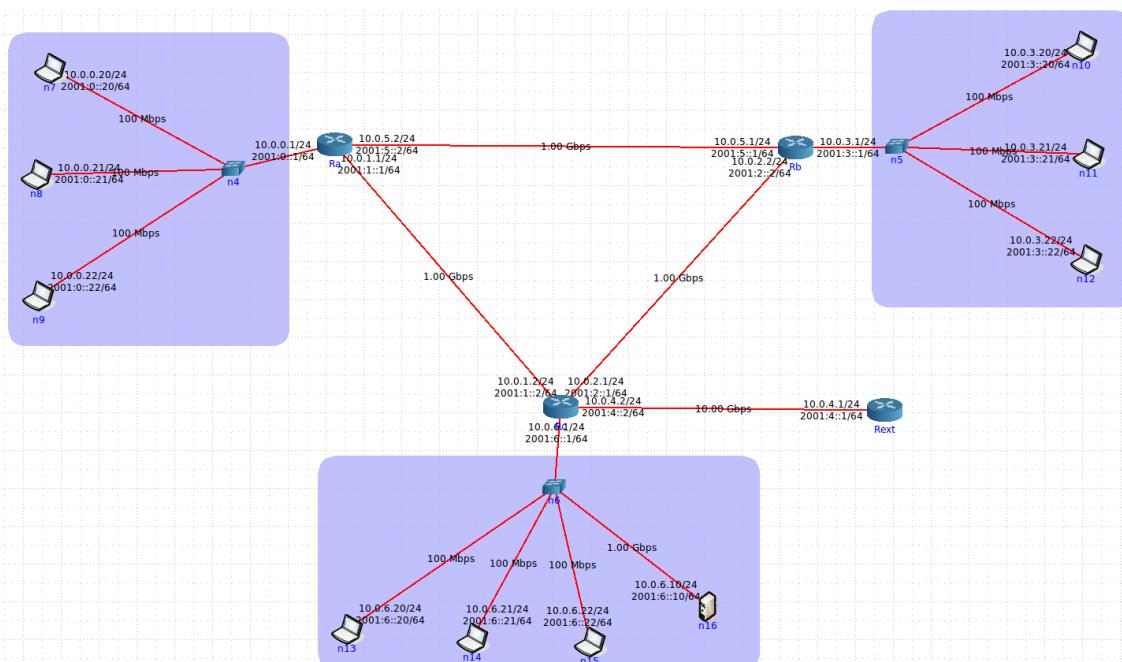


## 2<sup>a</sup>PART

### 1) a. Indique que endereços IP e máscaras de rede foram atribuídos pelo CORE a cada equipamento.

A imagem seguinte ilustra os vários endereços IP e a respetiva máscara (255.255.255.0, ou /24 em notação CIDR) atribuídos pelo CORE a cada equipamento.

Nota: O servidor n16 representa aquele que seria o “s1” mas, por lapso nosso, o nome ficou por alterar. Portanto, sempre que for referido n16 de agora em diante, note-se que nos referimos a s1.



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

Tratam-se de endereços privados, pois pertencem à Classe A (endereços entre 10.0.0.0 e 10.255.255.255).

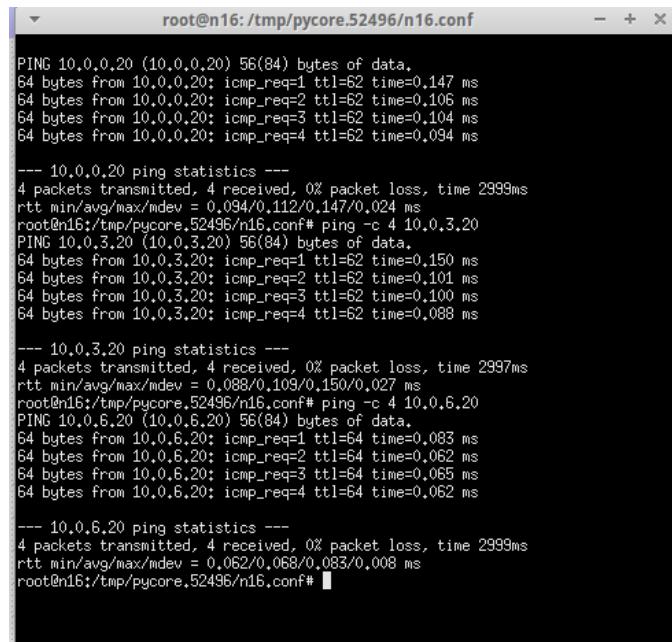
**c. Porque razão não é atribuído um endereço IP aos switches?**

O switch tem como principal função a interligação de equipamentos. Fazendo o registo do endereço MAC de cada um dos dispositivos que estão ligados a ele, consegue fazer o redireccionamento de pacotes entre esses mesmos equipamentos, com base na informação da sua própria tabela MAC.

A título de exemplo, consideremos um PC1, um PC2 e um PC3, todos ligados a um switch S. Inicialmente, S não sabe nada sobre os equipamentos que estão ligados a si. Se o PC1 envia informação para o PC3, o switch, não conhecendo a localização deste último, envia a informação a PC2 e PC3. O dispositivo PC3 “informa” o switch de que ele é o destinatário da informação e S regista o endereço MAC de PC1 e PC3 para que, numa próxima tentativa de comunicação entre os dois, o switch consiga encaminhar a informação diretamente ao destino.

**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).**

Através do resultado da execução do comando ping, podemos concluir que existe conectividade IP entre os laptops dos vários departamentos e o servidor do departamento C, pois os pacotes enviados a partir de S1 conseguem sempre chegar aos laptops das restantes redes.



```
root@n16:/tmp/pycore.52496/n16.conf
PING 10.0.0.20 (10.0.0.20) 56(84) bytes of data.
64 bytes from 10.0.0.20: icmp_req=1 ttl=62 time=0.147 ms
64 bytes from 10.0.0.20: icmp_req=2 ttl=62 time=0.106 ms
64 bytes from 10.0.0.20: icmp_req=3 ttl=62 time=0.104 ms
64 bytes from 10.0.0.20: icmp_req=4 ttl=62 time=0.094 ms
--- 10.0.0.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.094/0.112/0.147/0.024 ms
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 10.0.3.20
PING 10.0.3.20 (10.0.3.20) 56(84) bytes of data.
64 bytes from 10.0.3.20: icmp_req=1 ttl=62 time=0.150 ms
64 bytes from 10.0.3.20: icmp_req=2 ttl=62 time=0.101 ms
64 bytes from 10.0.3.20: icmp_req=3 ttl=62 time=0.100 ms
64 bytes from 10.0.3.20: icmp_req=4 ttl=62 time=0.088 ms
--- 10.0.3.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.088/0.109/0.150/0.027 ms
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 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_req=1 ttl=64 time=0.083 ms
64 bytes from 10.0.6.20: icmp_req=2 ttl=64 time=0.062 ms
64 bytes from 10.0.6.20: icmp_req=3 ttl=64 time=0.065 ms
64 bytes from 10.0.6.20: icmp_req=4 ttl=64 time=0.062 ms
--- 10.0.6.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.062/0.068/0.083/0.008 ms
root@n16:/tmp/pycore.52496/n16.conf#
```

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

Tal como na alínea anterior, vemos que é possível transferir pacotes entre S1 e Rext, provando assim que existe conectividade IP entre os dois.

```

root@n16:/tmp/pycore.52496/n16.conf
--- 10.0.6.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.062/0.068/0.083/0.008 ms
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 10.0.4.1
PING 10.0.4.1 (10.0.4.1) 56(84) bytes of data.
64 bytes from 10.0.4.1: icmp_req=1 ttl=63 time=0.064 ms
64 bytes from 10.0.4.1: icmp_req=2 ttl=63 time=0.099 ms
64 bytes from 10.0.4.1: icmp_req=3 ttl=63 time=0.102 ms
64 bytes from 10.0.4.1: icmp_req=4 ttl=63 time=0.100 ms

--- 10.0.4.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.064/0.091/0.102/0.017 ms
root@n16:/tmp/pycore.52496/n16.conf#

```

## 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).

Nas tabelas obtidas, temos alguma informação relativa à rota que o pacote terá de fazer. A coluna “Destination” indica a sub-rede de destino, a “Gateway” indica por que equipamento terá de passar o pacote e a “Genmask” o tipo de máscara usada.

No caso do router Ra, vemos que um pacote que lá esteja e que tenha como destino um equipamento da sub-rede 10.0.2.0 terá que passar pelo router 10.0.1.2.

Se o endereço de destino for 10.0.0.0, então o pacote segue um caminho qualquer, não tendo nenhum router específico como destino.

```

root@Ra:/tmp/pycore.52496/Ra.conf
root@Ra:/tmp/pycore.52496/Ra.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0         0.0.0.0        255.255.255.0    U        0 0          0 eth0
10.0.1.0         0.0.0.0        255.255.255.0    U        0 0          0 eth1
10.0.2.0         10.0.1.2       255.255.255.0    UG       0 0          0 eth1
10.0.3.0         10.0.5.1       255.255.255.0    UG       0 0          0 eth2
10.0.4.0         10.0.1.2       255.255.255.0    UG       0 0          0 eth1
10.0.5.0         0.0.0.0        255.255.255.0    U        0 0          0 eth2
10.0.6.0         10.0.1.2       255.255.255.0    UG       0 0          0 eth1
root@Ra:/tmp/pycore.52496/Ra.conf#

```

No caso do laptop n7 da rede do Departamento A, existem apenas duas rotas. A default, que indica que independentemente do endereço de destino, o pacote irá sempre para o router Ra, e uma que especifica o caminho a seguir quando o pacote de destino tem o endereço da sub-rede do próprio departamento, que é optar por um destino qualquer.

```

root@n7:/tmp/pycore.52496/n7.conf
root@n7:/tmp/pycore.52496/n7.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         10.0.0.1       0.0.0.0       UG        0 0          0 eth0
10.0.0.0        0.0.0.0        255.255.255.0 U          0 0          0 eth0
root@n7:/tmp/pycore.52496/n7.conf#

```

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

No router, está a ser usado encaminhamento dinâmico, pois executando o comando “ps -A” (que nos permite analisar os processos e protocolos em execução) verificamos que está a correr o protocolo ospfd. Este protocolo permite ao pacote seguir caminhos distintos quando não lhe é possível fazer a rota predefinida/esperada.

Pelo contrário, no laptop, o encaminhamento é estático. Ou o pacote segue as rotas que estão definidas na tabela de endereçamento, ou é descartado.

```

root@Ra:/tmp/pycore.52496/Ra.conf
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        0.0.0.0        255.255.255.0 U        0 0          0 eth0
10.0.1.0        0.0.0.0        255.255.255.0 U        0 0          0 eth1
10.0.2.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.3.0        10.0.5.1       255.255.255.0 UG       0 0          0 eth2
10.0.4.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.5.0        0.0.0.0        255.255.255.0 U        0 0          0 eth2
10.0.6.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
root@Ra:/tmp/pycore.52496/Ra.conf# ps
  PID TTY      TIME CMD
    77 pts/8    00:00:00 bash
  132 pts/8    00:00:00 ps
root@Ra:/tmp/pycore.52496/Ra.conf# ps -A
  PID TTY      TIME CMD
      1 ?        00:00:00 vnodesd
      42 ?        00:00:00 zebra
      71 ?        00:00:00 ospf6d
      72 ?        00:00:00 ospfd
    77 pts/8    00:00:00 bash
  133 pts/8    00:00:00 ps
root@Ra:/tmp/pycore.52496/Ra.conf#

```

```

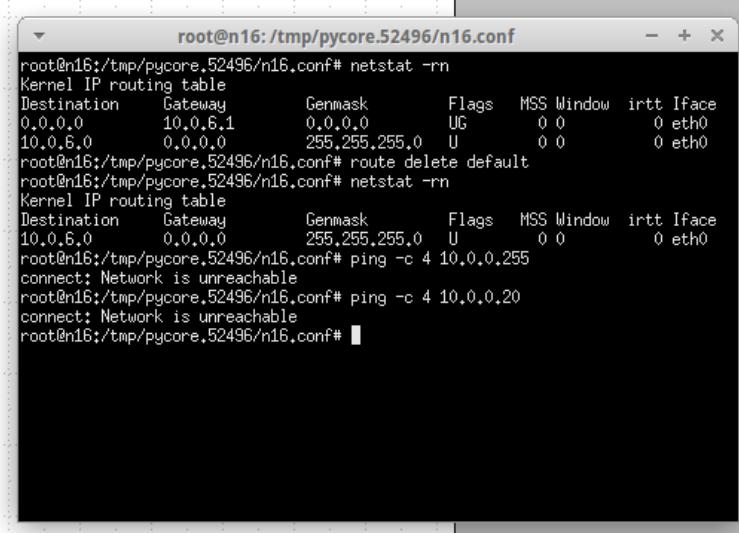
root@n7:/tmp/pycore.52496/n7.conf
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         10.0.0.1       0.0.0.0       UG        0 0          0 eth0
10.0.0.0        0.0.0.0        255.255.255.0 U          0 0          0 eth0
root@n7:/tmp/pycore.52496/n7.conf# ps
  PID TTY      TIME CMD
    72 pts/6    00:00:00 bash
  127 pts/6    00:00:00 ps
root@n7:/tmp/pycore.52496/n7.conf# ps -A
  PID TTY      TIME CMD
      1 ?        00:00:00 vnodesd
      72 pts/6    00:00:00 bash
    128 pts/6    00:00:00 ps
root@n7:/tmp/pycore.52496/n7.conf#

```

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.

Ao eliminar a rota por defeito, impossibilitamos o servidor S1 de efetuar ligações fora da rede do seu departamento, ou seja, este só consegue conectar-se aos equipamentos da sua rede. Numa tentativa de fazer ping para laptops de outros departamentos, recebemos a mensagem “Network is unreachable”.



```
root@n16:/tmp/pycore.52496/n16.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         10.0.6.1       0.0.0.0        UG      0 0          0 eth0
10.0.6.0        0.0.0.0       255.255.255.0  U        0 0          0 eth0
root@n16:/tmp/pycore.52496/n16.conf# route delete default
root@n16:/tmp/pycore.52496/n16.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.6.0        0.0.0.0       255.255.255.0  U        0 0          0 eth0
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 10.0.0.255
connect: Network is unreachable
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 10.0.0.20
connect: Network is unreachable
root@n16:/tmp/pycore.52496/n16.conf#
```

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.

```

root@n16:/tmp/pycore.52496/n16.conf - + ×
root@n16:/tmp/pycore.52496/n16.conf# route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.6.1
root@n16:/tmp/pycore.52496/n16.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        10.0.6.1       255.255.255.0  UG      0 0          0 eth0
10.0.6.0        0.0.0.0        255.255.255.0  U        0 0          0 eth0
root@n16:/tmp/pycore.52496/n16.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.6.1
root@n16:/tmp/pycore.52496/n16.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        10.0.6.1       255.255.255.0  UG      0 0          0 eth0
10.0.3.0        10.0.6.1       255.255.255.0  UG      0 0          0 eth0
10.0.6.0        0.0.0.0        255.255.255.0  U        0 0          0 eth0
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 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_req=1 ttl=62 time=0.147 ms
64 bytes from 10.0.0.20: icmp_req=2 ttl=62 time=0.149 ms
64 bytes from 10.0.0.20: icmp_req=3 ttl=62 time=0.090 ms
64 bytes from 10.0.0.20: icmp_req=4 ttl=62 time=0.105 ms
--- 10.0.0.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.090/0.122/0.149/0.029 ms
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 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_req=1 ttl=64 time=0.087 ms
64 bytes from 10.0.6.20: icmp_req=2 ttl=64 time=0.066 ms
64 bytes from 10.0.6.20: icmp_req=3 ttl=64 time=0.066 ms
64 bytes from 10.0.6.20: icmp_req=4 ttl=64 time=0.067 ms
--- 10.0.6.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.066/0.071/0.0877/0.012 ms
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 10.0.3.20
PING 10.0.3.20 (10.0.3.20) 56(84) bytes of data.
64 bytes from 10.0.3.20: icmp_req=1 ttl=62 time=0.127 ms
64 bytes from 10.0.3.20: icmp_req=2 ttl=62 time=0.055 ms
64 bytes from 10.0.3.20: icmp_req=3 ttl=62 time=0.089 ms
64 bytes from 10.0.3.20: icmp_req=4 ttl=62 time=0.107 ms
--- 10.0.3.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.055/0.094/0.127/0.028 ms
root@n16:/tmp/pycore.52496/n16.conf#

```

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.

Na imagem que se segue é possível verificar que o servidor S1 consegue aceder ao router Rext. Para além disso, na imagem apresentada na alínea anterior é possível verificar que também existe conexão entre S1 e as redes dos restantes departamentos.

```

root@n16:/tmp/pycore.52496/n16.conf - + ×
root@n16:/tmp/pycore.52496/n16.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        10.0.6.1       255.255.255.0  UG      0 0          0 eth0
10.0.3.0        10.0.6.1       255.255.255.0  UG      0 0          0 eth0
10.0.4.0        10.0.6.1       255.255.255.0  UG      0 0          0 eth0
10.0.6.0        0.0.0.0        255.255.255.0  U        0 0          0 eth0
root@n16:/tmp/pycore.52496/n16.conf# ping -c 4 10.0.4.1
PING 10.0.4.1 (10.0.4.1) 56(84) bytes of data.
64 bytes from 10.0.4.1: icmp_req=1 ttl=63 time=0.113 ms
64 bytes from 10.0.4.1: icmp_req=2 ttl=63 time=0.080 ms
64 bytes from 10.0.4.1: icmp_req=3 ttl=63 time=0.078 ms
64 bytes from 10.0.4.1: icmp_req=4 ttl=63 time=0.081 ms
--- 10.0.4.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.078/0.088/0.113/0.014 ms
root@n16:/tmp/pycore.52496/n16.conf#

```

3)

1. Considere que dispõe apenas do endereço de rede IP 172.46.48.0/20. 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.

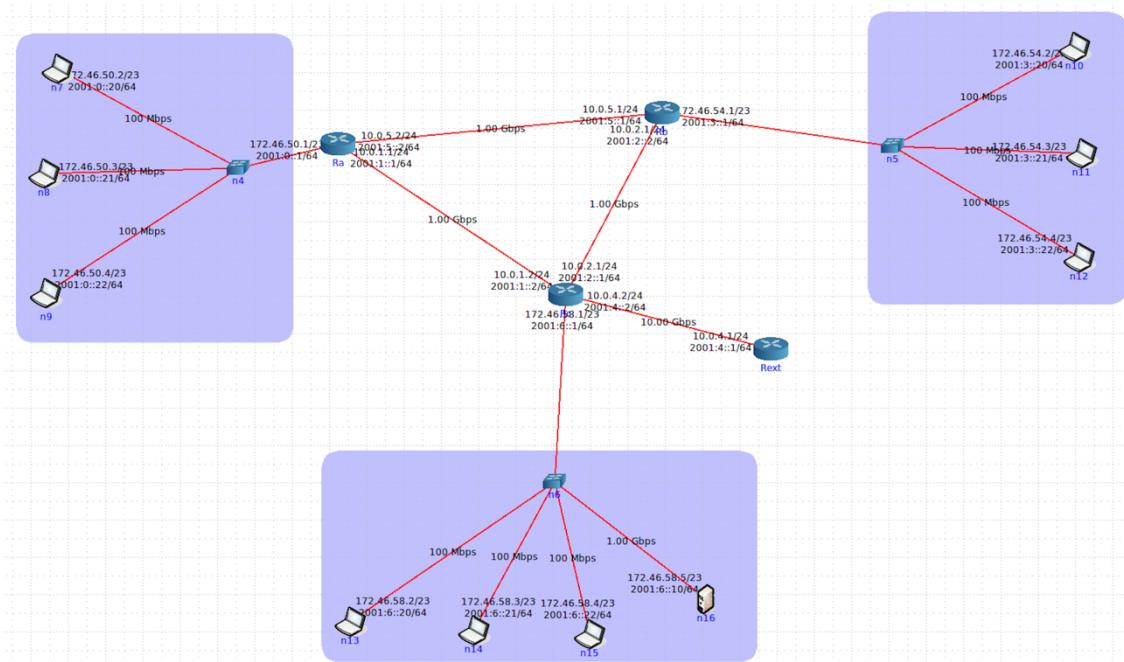
Dividindo o endereço de rede IP 172.46.48.0/20 conseguimos, à medida que andamos n bits para a direita, definir  $2^n - 2$  sub-redes. Para os 3 departamentos existentes, precisamos de definir 3 sub-redes. Para isto, optamos por usar 3 bits, ficando com  $(2^3 - 2) = 6$  endereços disponíveis.

A máscara de rede passa a ser 255.255.254.0 (/23).

Atribuímos a cada router o valor limite inferior da gama de endereços válidos de cada sub-rede. Na rede do departamento A (172.46.50.0/23), por exemplo, o router tem o endereço 172.46.50.1/23.

Os endereços atribuídos aos diferentes departamentos encontram-se na tabela seguinte:

<b>SR1</b>	000	Reservado (default)	-
<b>SR2</b>	001	172.46.50.0/23	Departamento A
<b>SR3</b>	010	172.46.52.0/23	livre
<b>SR4</b>	011	172.46.54.0/23	Departamento B
<b>SR5</b>	100	172.46.56.0/23	livre
<b>SR6</b>	101	172.46.58.0/23	Departamento C
<b>SR7</b>	110	172.46.60.0/23	livre
<b>SR8</b>	111	Reservado (broadcast)	-

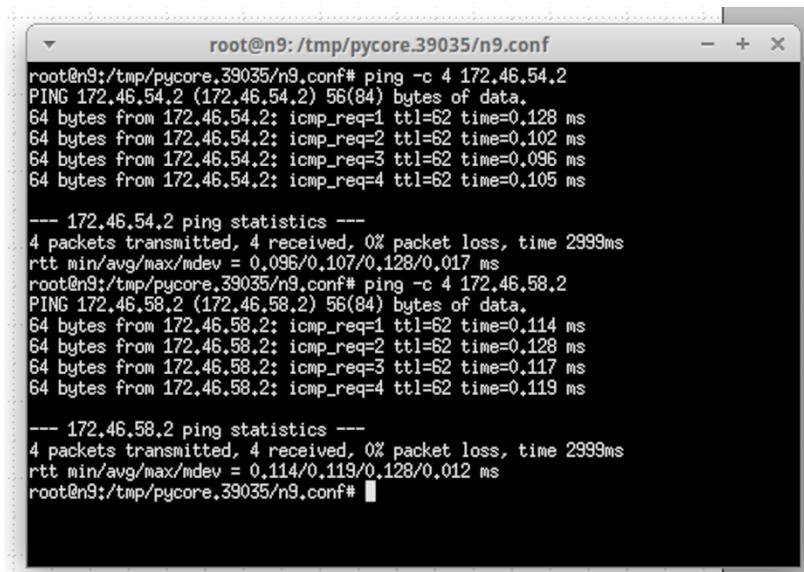


2. Qual a máscara de rede que usou (em formato decimal)? Quantos hosts IP pode interligar em cada departamento? Justifique.

A máscara de rede usada foi, em notação CIDR, /23, ou seja, usou-se a máscara 255.255.254.0. Em cada departamento é possível atribuir  $(2^{32-23}-2-1) = 509$  endereços de IP para hosts.

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

Fazendo *ping* de um laptop da rede A para laptops das outras redes, verificamos que existe conectividade IP entre as várias redes locais.



```
root@n9:/tmp/pycore.39035/n9.conf# ping -c 4 172.46.54.2
PING 172.46.54.2 (172.46.54.2) 56(84) bytes of data.
64 bytes from 172.46.54.2: icmp_req=1 ttl=62 time=0.128 ms
64 bytes from 172.46.54.2: icmp_req=2 ttl=62 time=0.102 ms
64 bytes from 172.46.54.2: icmp_req=3 ttl=62 time=0.096 ms
64 bytes from 172.46.54.2: icmp_req=4 ttl=62 time=0.105 ms

--- 172.46.54.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.096/0.107/0.128/0.017 ms
root@n9:/tmp/pycore.39035/n9.conf# ping -c 4 172.46.58.2
PING 172.46.58.2 (172.46.58.2) 56(84) bytes of data.
64 bytes from 172.46.58.2: icmp_req=1 ttl=62 time=0.114 ms
64 bytes from 172.46.58.2: icmp_req=2 ttl=62 time=0.128 ms
64 bytes from 172.46.58.2: icmp_req=3 ttl=62 time=0.117 ms
64 bytes from 172.46.58.2: icmp_req=4 ttl=62 time=0.119 ms

--- 172.46.58.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.114/0.119/0.128/0.012 ms
root@n9:/tmp/pycore.39035/n9.conf#
```

## CONCLUSÃO

Na primeira parte do trabalho, foi feita uma análise ao protocolo IPv4. Para esse efeito, realizamos uma topologia core, e analisamos o seu comportamento e o tráfego ICMP recebido. Para além disso, estudamos alguns casos particulares do IPv4, como a fragmentação de pacotes IP cujo tamanho é superior ao MTU da rede.

Na realização da segunda parte deste trabalho prático apuramos os nossos conhecimentos ao nível deste protocolo, nomeadamente no que toca ao endereçamento e encaminhamento IP. Mais especificamente, vimos, numa topologia core, como funciona o encaminhamento entre 3 redes diferentes, e como é feita a manipulação de endereços IP para efeitos de subnetting, e o seu encaminhamento respetivamente.