

UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

## Trabalho Prático 2

### Redes de Computadores

Grupo 21

Ana Filipa Pereira (A89589)      Carolina Santejo (A89500)  
Raquel Costa (A89464)

25 de novembro de 2020

## Conteúdo

<b>I</b>	<b>TP2: Protocolo IPv4</b>	<b>4</b>
<b>1</b>	<b>Exercício 1</b>	<b>4</b>
1.1	a) . . . . .	4
1.2	b) . . . . .	5
1.3	c) . . . . .	5
1.4	d) . . . . .	6
<b>2</b>	<b>Exercício 2</b>	<b>6</b>
2.1	a) . . . . .	6
2.2	b) . . . . .	6
2.3	c) . . . . .	6
2.4	d) . . . . .	6
2.5	e) . . . . .	6
2.6	f) . . . . .	7
2.7	g) . . . . .	7
<b>3</b>	<b>Exercício 3</b>	<b>8</b>
3.1	a) . . . . .	8
3.2	b) . . . . .	8
3.3	c) . . . . .	9
3.4	d) . . . . .	9
3.5	e) . . . . .	9
<b>II</b>	<b>TP2: Protocolo IPv4</b>	<b>11</b>
<b>1</b>	<b>Exercício 1</b>	<b>11</b>
1.1	a) . . . . .	11
1.2	b) . . . . .	11
1.3	c) . . . . .	11
1.4	d) . . . . .	11
1.5	e) . . . . .	12
<b>2</b>	<b>Exercício 2</b>	<b>13</b>
2.1	a) . . . . .	13
2.2	b) . . . . .	13
2.3	c) . . . . .	13
2.4	d) . . . . .	14
2.5	e) . . . . .	15

<b>3</b>	<b>Exercício 3</b>	<b>16</b>
3.1	a) . . . . .	16
3.2	b) . . . . .	16
3.3	c) . . . . .	17
	<b>Conclusão</b>	<b>21</b>

## Parte I

# TP2: Protocolo IPv4

## 1 Exercício 1

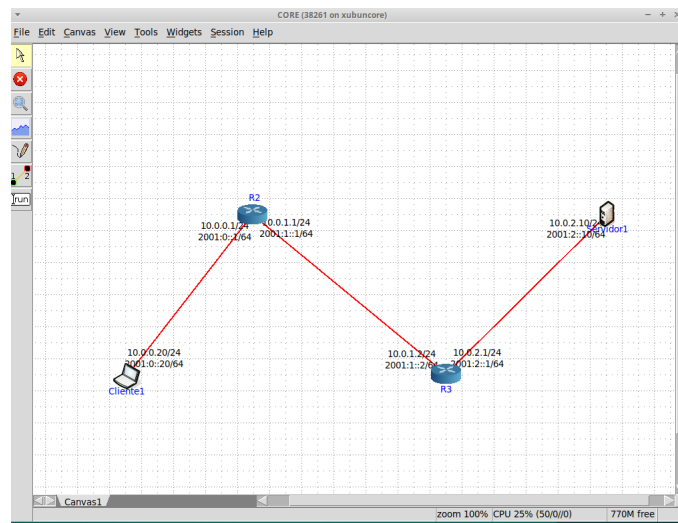


Figura 1: Topologia Core

**1.1 a)** Active o *wireshark* ou o *tcpdump* no *Cliente1*. Numa shell do *Cliente1*, execute o comando *traceroute -I* para o endereço IP do *Servidor1*.

```
root@Cliente1:/tmp/pycore.38261/Cliente1.conf# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.087 ms 0.022 ms 0.016 ms
 2 10.0.1.2 (10.0.1.2) 0.048 ms 0.023 ms 0.023 ms
 3 10.0.2.10 (10.0.2.10) 0.092 ms 0.030 ms 0.055 ms
root@Cliente1:/tmp/pycore.38261/Cliente1.conf# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 0.088 ms 0.023 ms 0.017 ms
 2 10.0.1.2 (10.0.1.2) 0.038 ms 0.025 ms 0.023 ms
 3 10.0.2.10 (10.0.2.10) 0.042 ms 0.031 ms 0.028 ms
root@Cliente1:/tmp/pycore.38261/Cliente1.conf#
```

Figura 2: Execução do comando *traceroute -I*

**1.2 b) Registe e analise o tráfego ICMP enviado pelo *Cliente1* e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.**

No.	Time	Source	Destination	Protocol	Length	Info
3	10.007554327	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
4	10.428685331	fe80::200:ff:feaa:1	ff02::5	OSPF	90	Hello Packet
5	20.011554322	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
6	20.430427851	fe80::200:ff:feaa:1	ff02::5	OSPF	90	Hello Packet
7	30.012357939	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
8	30.448156891	fe80::200:ff:feaa:1	ff02::5	OSPF	90	Hello Packet
9	34.951840586	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=1/256, ttl=1 (no response found!)
10	34.951888515	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
11	34.951916993	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=2/512, ttl=1 (no response found!)
12	34.951924885	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
13	34.951941156	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=3/768, ttl=1 (no response found!)
14	34.951955872	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
15	34.951969997	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=4/1024, ttl=2 (no response found!)
16	34.952000000	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
17	34.952014583	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=5/1280, ttl=2 (no response found!)
18	34.952031958	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
19	34.952045210	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=6/1536, ttl=2 (no response found!)
20	34.952060001	10.0.0.1	10.0.0.20	ICMP	102	Time to live exceeded (Time to live exceeded in transit)
21	34.952075568	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=7/1792, ttl=3 (reply in 22)
22	34.952111471	10.0.2.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0039, seq=7/1792, ttl=62 (request in 21)
23	34.952128365	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request id=0x0039, seq=8/2048, ttl=3 (reply in 24)
24	34.952151142	10.0.2.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0039, seq=8/2048, ttl=62 (request in 23)

Figura 3: Análise do tráfego ICMP

O comportamento observado é o esperado, pois são enviados, a partir do *Cliente1*, vários pacotes *Echo (Ping)Request*. Todos estes packages que são enviados com TTL igual a 1 ou 2 são descartados pelo router 2 e o router 3, respetivamente. Isto acontece, pois nestes end-systems o TTL desses pacotes atinge o valor zero. Deste modo, o router que descartou um pacote envia ao cliente 1 uma mensagem de controlo indicando que o TTL foi excedido (Time to live Exceeded). Os únicos pacotes que atingem o destino, o servidor 1, são os que possuem TTL=3 (Echo ping reply). Isto pois, cada vez que passa por um router o TTL decrementa em uma unidade. Desta forma, visto que a nossa possui 2 routers, só pacotes com TTL igual ou superior a 3 atingem o destino, como se verifica na figura 3.

**1.3 c) Qual deve ser o valor inicial mínimo do campo TTL para alcançar o *Servidor1*? Verifique na prática que a sua resposta está correta.**

21	34.952075568	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request	id=0x0039, seq=7/1792, ttl=3 (reply in 22)
22	34.952111471	10.0.2.10	10.0.0.20	ICMP	74	Echo (ping) reply	id=0x0039, seq=7/1792, ttl=62 (request in 21)
23	34.952128365	10.0.0.20	10.0.2.10	ICMP	74	Echo (ping) request	id=0x0039, seq=8/2048, ttl=3 (reply in 24)
24	34.952151142	10.0.2.10	10.0.0.20	ICMP	74	Echo (ping) reply	id=0x0039, seq=8/2048, ttl=62 (request in 23)

Figura 4: Valor mínimo TTL

O valor inicial mínimo é 3.

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

$$m = (0.042 + 0.031 + 0.028)/3 = 0.03367 \text{ ms}$$

## 2 Exercício 2

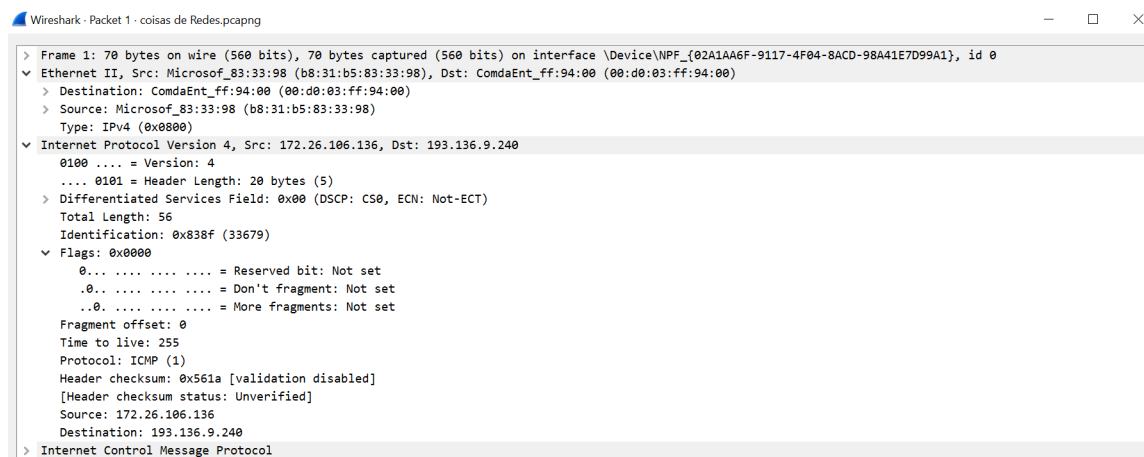


Figura 5: Análise de um pacote

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

O endereço IP é 172.26.106.136.

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

O valor do campo protocolo é ICMP (1) e significa *Internet Protocol*.

**2.3 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 tamanho do payload é o tamanho total do pacote menos o tamanho do cabeçalho.

Header (cabeçalho) = 20 Bytes

Payload = 56 - 20 = 36 Bytes

**2.4 d) O datagrama IP foi fragmentado? Justifique.**

Não está fragmentado pois o *Fragment Offset* é igual a 0 e o *More fragments* está definido como *Not set*, ou seja, não existem mais fragmentos para além deste.

**2.5 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
1	0.000000	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1726/48646, ttl=255 (reply in 2)
3	0.049912	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1727/48902, ttl=1 (no response found)
5	0.100547	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1728/49158, ttl=2 (no response found)
7	0.151005	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1729/49414, ttl=3 (no response found)
9	0.201556	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1730/49670, ttl=4 (reply in 10)
11	2.500212	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1731/49926, ttl=255 (reply in 12)
13	2.550193	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1732/50182, ttl=1 (no response found)
15	2.600105	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1733/50438, ttl=2 (no response found)
17	2.651298	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1734/50694, ttl=3 (no response found)
19	2.701868	172.26.106.136	193.136.9.240	ICMP	70	Echo (ping) request id=0x0001, seq=1735/50950, ttl=4 (reply in 20)
21	3.946708	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=1 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
22	3.946708	172.26.106.136	13.107.42.12	TLSv1	608	Application Data
23	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=1805 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
24	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=3055 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
25	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=4305 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
26	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=5555 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
27	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=6805 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
28	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=8055 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
29	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=9305 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
30	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=10555 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
31	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=11805 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
32	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=13055 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
33	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=14305 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
34	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=15555 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
35	3.947174	172.26.106.136	13.107.42.12	TLSv1	608	Application Data [TCP segment of a reassembled PDU]
36	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=16805 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
37	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=18055 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
38	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=19305 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
39	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=20555 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
40	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=21805 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
41	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=23055 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
42	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=24305 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
43	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=25555 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
44	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=26805 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
45	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=28055 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
46	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=29305 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
47	3.947174	172.26.106.136	13.107.42.12	TLSv1	608	Application Data [TCP segment of a reassembled PDU]
48	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=30555 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
49	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=31805 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]
50	3.947174	172.26.106.136	13.107.42.12	TCP	1304	50583 → 443 [ACK] Seq=33055 Ack=1 Win=508 Len=1250 [TCP segment of a reassembled PDU]

Figura 6: Tráfego capturado pelo wireshark

Os campos do cabeçalho IP que variam de pacote para pacote são o *Header checksum*, *identification* e o TTL

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

Sim. A identificação mantém-se constante (igual a 0x0001) e o TTL incrementa.

## 2.7 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?

No.	Time	Source	Destination	Protocol	Length	Info
2	0.008599	193.136.9.240	172.26.106.136	ICMP	70	Echo (ping) reply id=0x0001, seq=1726/48646, ttl=61 (request in 1)
4	0.053347	172.26.254.254	172.26.106.136	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
6	0.102554	172.16.2.1	172.26.106.136	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
8	0.153814	172.16.115.252	172.26.106.136	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10	0.283299	193.136.9.240	172.26.106.136	ICMP	70	Echo (ping) reply id=0x0001, seq=1730/49678, ttl=61 (request in 9)
12	2.508975	193.136.9.240	172.26.106.136	ICMP	70	Echo (ping) reply id=0x0001, seq=1731/49926, ttl=61 (request in 11)
14	2.552374	172.26.254.254	172.26.106.136	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
16	2.604804	172.16.2.1	172.26.106.136	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
18	2.654179	172.16.115.252	172.26.106.136	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
20	2.705811	193.136.9.240	172.26.106.136	ICMP	70	Echo (ping) reply id=0x0001, seq=1735/50950, ttl=61 (request in 19)

Figura 7: Tráfego capturado ordenado por endereço destino

O valor do TTL é 255 e não se mantém constante. Isto acontece pois o host envia um pacote com TTL=1. Este é descartado pelo router 1 (pois o TTL do pacote atingiu o valor zero) que envia á origem uma mensagem de erro (“TTL exceeded”). Esta mensagem é enviada com TTL=255, de forma a garantir que chega ao host. Visto que entre o host e o router 1 não existe outro router, o TTL da mensagem não decrementa. De seguida, o host envia um novo pacote com TTL=2 sendo que este decrementa no router 1 e atinge o valor zero quando chega ao router 2. Ora, o router 2 envia também ao host a mensagem “TTL exceeded” com TTL= 255 que diminui para 254 quando passa no router 1. O host envia outro pacote, mas com TTL = 3 , que decrementa no router 1 e 2 e atinge o valor zero no router 3. Este envia á origem a mensagem “TTL exceeded” com também TTL=255 que atinge o valor 253 (pois decrementou no router 1 e 2). No entanto, mensagens com TTL = 4 já atingem o destino, logo podemos concluir que existem 3 routers entre a origem e o destino final.

### 3 Exercício 3

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.26.16.43	193.136.9.240	ICMP	1514	Echo (ping) request id=0x0001, seq=31/7936, ttl=255 (reply in 4)
2	0.000000	172.26.16.43	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=f0b9)
3	0.000000	172.26.16.43	193.136.9.240	IPv4	275	Fragmented IP protocol (proto=ICMP 1, off=2960, ID=f0b9)

Figura 8: Mensagem fragmentada capturada pelo wireshark

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

A mensagem foi fragmentada porque o tamanho do pacote é maior do que o permitido.

#### 3.2 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?



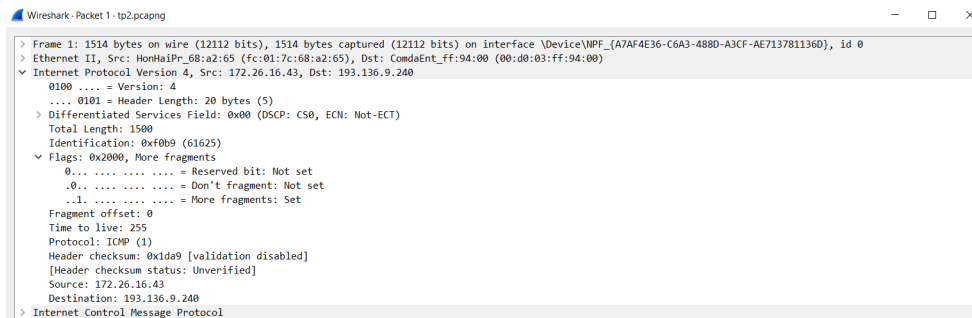


Figura 9: Primeiro fragmento do datagrama IP

Como se pode ver na figura trata-se do primeiro fragmento do datagrama pois a flag *More fragments* está caracterizada como 'Set' logo existem mais fragmentos e o *fragment offset* é 0 logo está na primeira posição do datagrama.

### 3.3 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?

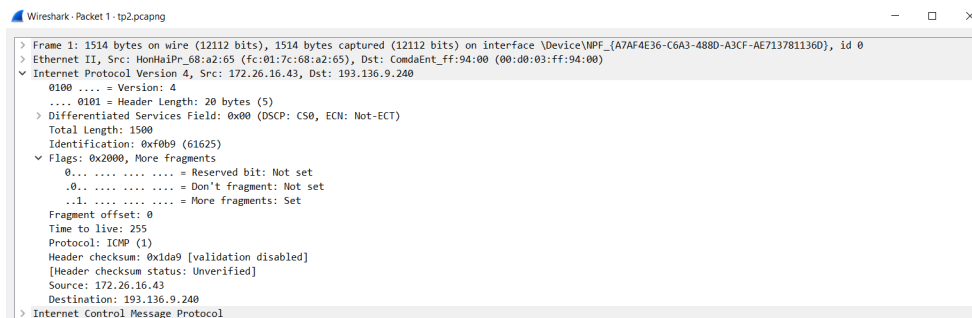


Figura 10: Primeiro fragmento do datagrama IP

Não se trata do primeiro fragmento pois o *fragment offset* é diferente de 0 e existem mais fragmentos porque a flag *More fragments* está caracterizada como 'Set'.

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

Foram criados 3 fragmentos a partir do datagrama original. Para detetar o ultimo é necessário encontrar o fragmento em que a flag *More fragments* está caracterizada como 'Not set' e o *fragment offset* é diferente de 0.

### 3.5 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 campos que variam são o tamanho, as flags, o *fragment offset* e o *Header checksum*. Estas informações permitem reconstruir o datagrama original uma vez que, a partir do tamanho sabe-se quantos bits tem cada fragmento, com as flags sabe-se se é necessário acrescentar mais algum fragmento, o *fragment offset* indica a posição do fragmento no datagrama e o *Header checksum* valida se o conteúdo da mensagem não foi corrompido

## Parte II

# TP2: Protocolo IPv4

## 1 Exercício 1

**1.1 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.

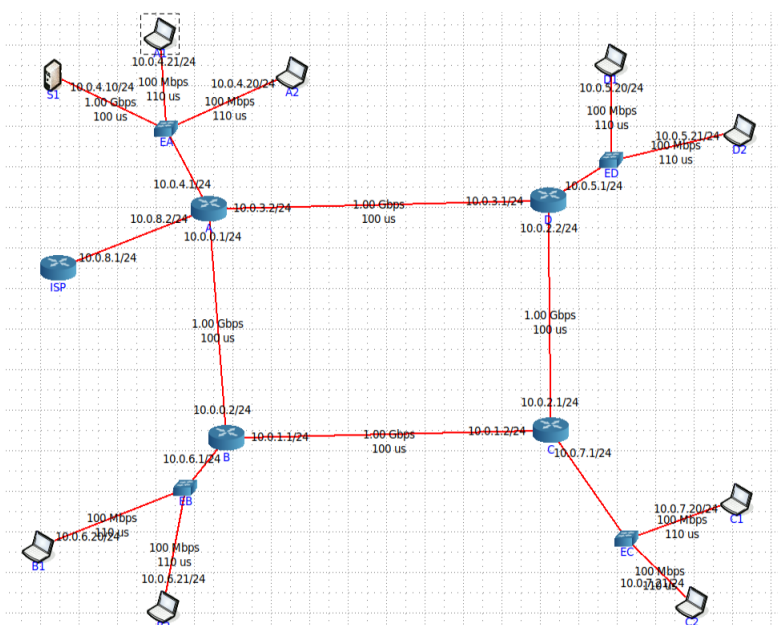


Figura 11: Topologia CORE

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

Trata-se de endereços privados porque estão dentro da faixa de endereços IP (10.0.0.0 – 10.255.255.255) reservada a endereços privados na norma RFC 1918.

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

Não foi atribuído endereço IP aos switches porque estes operam numa camada abaixo (Layer 2) e nesta camada não se comunica via IP.

**1.4 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).

```
root@A1:/tmp/pycore.44897/A1.conf# ping 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_seq=1 ttl=64 time=0.515 ms
64 bytes from 10.0.4.1: icmp_seq=2 ttl=64 time=0.582 ms
64 bytes from 10.0.4.1: icmp_seq=3 ttl=64 time=0.510 ms
64 bytes from 10.0.4.1: icmp_seq=4 ttl=64 time=0.487 ms
64 bytes from 10.0.4.1: icmp_seq=5 ttl=64 time=0.232 ms
64 bytes from 10.0.4.1: icmp_seq=6 ttl=64 time=0.138 ms
64 bytes from 10.0.4.1: icmp_seq=7 ttl=64 time=0.381 ms
64 bytes from 10.0.4.1: icmp_seq=8 ttl=64 time=1.02 ms
64 bytes from 10.0.4.1: icmp_seq=9 ttl=64 time=0.153 ms
64 bytes from 10.0.4.1: icmp_seq=10 ttl=64 time=0.156 ms
64 bytes from 10.0.4.1: icmp_seq=11 ttl=64 time=0.228 ms
64 bytes from 10.0.4.1: icmp_seq=12 ttl=64 time=0.158 ms
64 bytes from 10.0.4.1: icmp_seq=13 ttl=64 time=0.276 ms
64 bytes from 10.0.4.1: icmp_seq=14 ttl=64 time=0.415 ms
64 bytes from 10.0.4.1: icmp_seq=15 ttl=64 time=0.157 ms
64 bytes from 10.0.4.1: icmp_seq=16 ttl=64 time=0.533 ms
64 bytes from 10.0.4.1: icmp_seq=17 ttl=64 time=0.275 ms
^C
--- 10.0.4.1 ping statistics ---
17 packets transmitted, 17 received, 0% packet loss, time 1783ms
rtt min/avg/max/ndev = 0.153/0.338/1.023/0.269 ms
root@B1:/tmp/pycore.44897/B1.conf# []

root@D1:/tmp/pycore.44897/D1.conf# ping 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_seq=1 ttl=63 time=2.54 ms
64 bytes from 10.0.4.1: icmp_seq=2 ttl=63 time=2.03 ms
64 bytes from 10.0.4.1: icmp_seq=3 ttl=63 time=2.34 ms
64 bytes from 10.0.4.1: icmp_seq=4 ttl=63 time=1.59 ms
64 bytes from 10.0.4.1: icmp_seq=5 ttl=63 time=2.02 ms
64 bytes from 10.0.4.1: icmp_seq=6 ttl=63 time=1.31 ms
64 bytes from 10.0.4.1: icmp_seq=7 ttl=63 time=0.449 ms
64 bytes from 10.0.4.1: icmp_seq=8 ttl=63 time=1.40 ms
64 bytes from 10.0.4.1: icmp_seq=9 ttl=63 time=1.23 ms
64 bytes from 10.0.4.1: icmp_seq=10 ttl=63 time=2.31 ms
64 bytes from 10.0.4.1: icmp_seq=11 ttl=63 time=1.39 ms
64 bytes from 10.0.4.1: icmp_seq=12 ttl=63 time=1.34 ms
64 bytes from 10.0.4.1: icmp_seq=13 ttl=63 time=0.729 ms
64 bytes from 10.0.4.1: icmp_seq=14 ttl=63 time=1.53 ms
64 bytes from 10.0.4.1: icmp_seq=15 ttl=63 time=1.27 ms
64 bytes from 10.0.4.1: icmp_seq=16 ttl=63 time=0.816 ms
64 bytes from 10.0.4.1: icmp_seq=17 ttl=63 time=1.77 ms
64 bytes from 10.0.4.1: icmp_seq=18 ttl=63 time=0.324 ms
64 bytes from 10.0.4.1: icmp_seq=19 ttl=63 time=0.382 ms
64 bytes from 10.0.4.1: icmp_seq=20 ttl=63 time=2.36 ms
^C
--- 10.0.4.1 ping statistics ---
21 packets transmitted, 21 received, 0% packet loss, time 2008ms
rtt min/avg/max/ndev = 0.449/1.527/4.738/0.873 ms
root@B1:/tmp/pycore.44897/B1.conf# []

root@C1:/tmp/pycore.44897/C1.conf# ping 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_seq=1 ttl=62 time=2.47 ms
64 bytes from 10.0.4.1: icmp_seq=2 ttl=62 time=2.04 ms
64 bytes from 10.0.4.1: icmp_seq=3 ttl=62 time=1.56 ms
64 bytes from 10.0.4.1: icmp_seq=4 ttl=62 time=3.03 ms
64 bytes from 10.0.4.1: icmp_seq=5 ttl=62 time=1.50 ms
64 bytes from 10.0.4.1: icmp_seq=6 ttl=62 time=1.33 ms
64 bytes from 10.0.4.1: icmp_seq=7 ttl=62 time=1.38 ms
64 bytes from 10.0.4.1: icmp_seq=8 ttl=62 time=3.09 ms
64 bytes from 10.0.4.1: icmp_seq=9 ttl=62 time=2.79 ms
64 bytes from 10.0.4.1: icmp_seq=10 ttl=62 time=1.53 ms
64 bytes from 10.0.4.1: icmp_seq=11 ttl=62 time=1.46 ms
64 bytes from 10.0.4.1: icmp_seq=12 ttl=62 time=3.11 ms
64 bytes from 10.0.4.1: icmp_seq=13 ttl=62 time=2.40 ms
64 bytes from 10.0.4.1: icmp_seq=14 ttl=62 time=1.88 ms
64 bytes from 10.0.4.1: icmp_seq=15 ttl=62 time=3.30 ms
64 bytes from 10.0.4.1: icmp_seq=16 ttl=62 time=2.36 ms
64 bytes from 10.0.4.1: icmp_seq=17 ttl=62 time=1.37 ms
64 bytes from 10.0.4.1: icmp_seq=18 ttl=62 time=1.45 ms
64 bytes from 10.0.4.1: icmp_seq=19 ttl=62 time=1.44 ms
64 bytes from 10.0.4.1: icmp_seq=20 ttl=62 time=1.38 ms
^C
--- 10.0.4.1 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 1902ms
rtt min/avg/max/ndev = 1.333/2.069/3.307/0.862 ms
root@B1:/tmp/pycore.44897/B1.conf# []

root@B1:/tmp/pycore.44897/B1.conf# ping 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_seq=1 ttl=63 time=2.86 ms
64 bytes from 10.0.4.1: icmp_seq=2 ttl=63 time=1.54 ms
64 bytes from 10.0.4.1: icmp_seq=3 ttl=63 time=2.12 ms
64 bytes from 10.0.4.1: icmp_seq=4 ttl=63 time=0.804 ms
64 bytes from 10.0.4.1: icmp_seq=5 ttl=63 time=2.31 ms
64 bytes from 10.0.4.1: icmp_seq=6 ttl=63 time=1.12 ms
64 bytes from 10.0.4.1: icmp_seq=7 ttl=63 time=1.04 ms
64 bytes from 10.0.4.1: icmp_seq=8 ttl=63 time=0.339 ms
64 bytes from 10.0.4.1: icmp_seq=9 ttl=63 time=2.27 ms
64 bytes from 10.0.4.1: icmp_seq=10 ttl=63 time=0.854 ms
64 bytes from 10.0.4.1: icmp_seq=11 ttl=63 time=1.37 ms
64 bytes from 10.0.4.1: icmp_seq=12 ttl=63 time=2.82 ms
64 bytes from 10.0.4.1: icmp_seq=13 ttl=63 time=2.25 ms
64 bytes from 10.0.4.1: icmp_seq=14 ttl=63 time=1.69 ms
64 bytes from 10.0.4.1: icmp_seq=15 ttl=63 time=1.43 ms
64 bytes from 10.0.4.1: icmp_seq=16 ttl=63 time=2.89 ms
^C
--- 10.0.4.1 ping statistics ---
16 packets transmitted, 16 received, 0% packet loss, time 1546ms
rtt min/avg/max/ndev = 0.504/1.817/2.325/0.722 ms
root@B1:/tmp/pycore.44897/B1.conf# []
```

Figura 12: Execução do comando *ping* nos Laptops A1, B1, C1 e D1

Como se pode verificar na figura, existe conexão do servidor com os Laptops A1,B1,C1 e D1, podendo assim comprovar a existência de conectividade com os vários departamentos.

## 1.5 e) Verifique se existe conectividade IP do router de acesso R-ISP para o servidor S1.

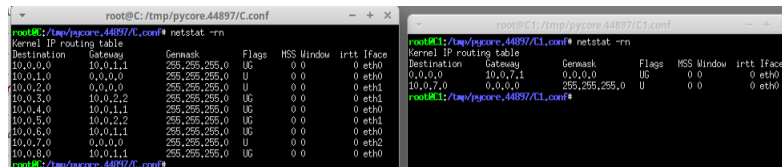
```
root@ISP:/tmp/pycore.44897/ISP.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data:
64 bytes from 10.0.4.10: icmp_seq=1 ttl=63 time=0.463 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=63 time=0.198 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=63 time=0.242 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=63 time=0.165 ms
64 bytes from 10.0.4.10: icmp_seq=5 ttl=63 time=0.230 ms
64 bytes from 10.0.4.10: icmp_seq=6 ttl=63 time=0.395 ms
64 bytes from 10.0.4.10: icmp_seq=7 ttl=63 time=0.168 ms
64 bytes from 10.0.4.10: icmp_seq=8 ttl=63 time=0.174 ms
64 bytes from 10.0.4.10: icmp_seq=9 ttl=63 time=0.328 ms
64 bytes from 10.0.4.10: icmp_seq=10 ttl=63 time=0.426 ms
64 bytes from 10.0.4.10: icmp_seq=11 ttl=63 time=1.03 ms
64 bytes from 10.0.4.10: icmp_seq=12 ttl=63 time=0.679 ms
64 bytes from 10.0.4.10: icmp_seq=13 ttl=63 time=0.488 ms
64 bytes from 10.0.4.10: icmp_seq=14 ttl=63 time=0.331 ms
64 bytes from 10.0.4.10: icmp_seq=15 ttl=63 time=0.422 ms
64 bytes from 10.0.4.10: icmp_seq=16 ttl=63 time=0.636 ms
64 bytes from 10.0.4.10: icmp_seq=17 ttl=63 time=0.195 ms
64 bytes from 10.0.4.10: icmp_seq=18 ttl=63 time=0.502 ms
^C
--- 10.0.4.10 ping statistics ---
18 packets transmitted, 18 received, 0% packet loss, time 1738ms
rtt min/avg/max/ndev = 0.155/0.402/1.032/0.280 ms
root@ISP:/tmp/pycore.44897/ISP.conf# []
```

Figura 13: Execução do comando *ping* no Router ISP

Como se pode verificar na figura, existe conexão do router ISP para o servidor S1.

## 2 Exercício 2

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



```
root@C:/tmp/pycore.44897/C.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 10.0.2.2 255.255.255.0 UG 0 0 0 eth1
10.0.6.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
10.0.8.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0

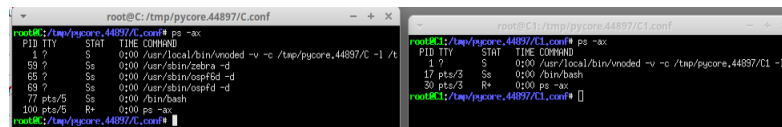
root@C1:/tmp/pycore.44897/C1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.7.1 0.0.0.0 UG 0 0 0 eth0
10.0.7.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@C1:/tmp/pycore.44897/C1.conf#
```

Figura 14: Execução do comando `netstat -rn` no Router C e no Laptop C1

Como se pode verificar pela figura acima, cada linha vai indicar que para cada endereço de destino (*Destination*), o datagrama vai ser inicialmente direcionado para o endereço da coluna *gateway* saindo pela interface local *Iface*. O valor da máscara é indicado pela coluna *Genmask*. As *Flags* servem para acrescentar informações adicionais, sendo U utilizada quando o gateway não está definido e UG caso contrário.

Considerando agora os valores das tabelas, verifica-se que algumas entradas têm *Gateway* 0.0.0.0, isto significa que o *Gateway* não está definido, ou seja, a ligação entre os dois endereços é direta.

**2.2 b)** Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema, por exemplo, `ps -ax`).



```
root@C:/tmp/pycore.44897/C.conf# ps -ax
PID TTY STAT TIME COMMAND
1 ? S 0:00 /usr/local/bin/vmnode -v -c /tmp/pycore.44897/C -1 /t
59 ? Ss 0:00 /usr/bin/strace -o
69 ? Ss 0:00 /usr/bin/ospfd -d
77 pts/5 Ss 0:00 /bin/bash
100 pts/5 R+ 0:00 ps -ax

root@C1:/tmp/pycore.44897/C1.conf# ps -ax
PID TTY STAT TIME COMMAND
1 ? S 0:00 /usr/local/bin/vmnode -v -c /tmp/pycore.44897/C1 -1 /
17 pts/3 Ss 0:00 /bin/bash
30 pts/3 R+ 0:00 ps -ax
root@C1:/tmp/pycore.44897/C1.conf#
```

Figura 15: Execução do comando `ps -ax` no Router C e no Laptop C1

Está a ser utilizado encaminhamento dinâmico, uma vez que, as rotas são definidas automaticamente através da troca de informação entre routers.

**2.3 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 implicações tem esta medida para os utilizadores da organização MIEI-RC que acedem ao servidor. Justifique.

```

root@S1: /tmp/pycore.44897/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.4.1 0.0.0.0 UG 0 0 0 eth0
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@S1: /tmp/pycore.44897/S1.conf#
root@S1: /tmp/pycore.44897/S1.conf#
root@S1: /tmp/pycore.44897/S1.conf# route delete default
root@S1: /tmp/pycore.44897/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@S1: /tmp/pycore.44897/S1.conf#

```

Figura 16: Execução do comando *route delete default* no servidor S1

A execução do comando *route delete* vai resultar na perda de conexão do servidor S1 com todos os hosts que não pertencem à sua rede local (Departamento A). Isto acontece porque foi eliminada a rota por defeito, ou seja, o servidor deixa de ter definida a rota de tráfego para redes não locais.

**2.4 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.**

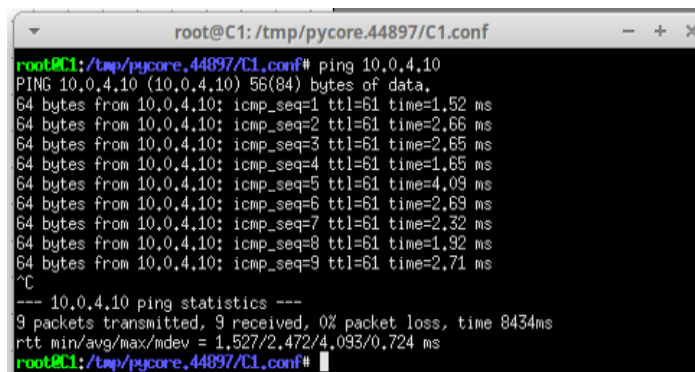
```

root@S1: /tmp/pycore.44897/S1.conf# route add -net 10.0.5.0/24 gw 10.0.3.2
SIOCADDRT: Network is unreachable
root@S1: /tmp/pycore.44897/S1.conf# route add -net 10.0.5.0/24 gw 10.0.4.1
root@S1: /tmp/pycore.44897/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.5.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
root@S1: /tmp/pycore.44897/S1.conf# route add -net 10.0.7.0/24 gw 10.0.4.1
root@S1: /tmp/pycore.44897/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.5.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
root@S1: /tmp/pycore.44897/S1.conf# route add -net 10.0.6.0/24 gw 10.0.4.1
root@S1: /tmp/pycore.44897/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.5.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.6.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
root@S1: /tmp/pycore.44897/S1.conf#

```

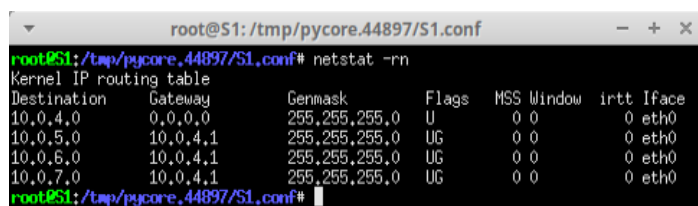
Figura 17: Adição de novas rotas estáticas

**2.5 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@C1: /tmp/pycore.44897/C1.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_seq=1 ttl=61 time=1.52 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=61 time=2.66 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=61 time=2.65 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=61 time=1.65 ms
64 bytes from 10.0.4.10: icmp_seq=5 ttl=61 time=4.09 ms
64 bytes from 10.0.4.10: icmp_seq=6 ttl=61 time=2.69 ms
64 bytes from 10.0.4.10: icmp_seq=7 ttl=61 time=2.32 ms
64 bytes from 10.0.4.10: icmp_seq=8 ttl=61 time=1.92 ms
64 bytes from 10.0.4.10: icmp_seq=9 ttl=61 time=2.71 ms
^C
--- 10.0.4.10 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8434ms
rtt min/avg/max/mdev = 1.527/2.472/4.093/0.724 ms
root@C1: /tmp/pycore.44897/C1.conf#
```

Figura 18: Execução do comando ping em S1



```
root@S1: /tmp/pycore.44897/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.5.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.6.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
root@S1: /tmp/pycore.44897/S1.conf#
```

Figura 19: Execução do comando `netstat -rn` no servidor S1

### 3 Exercício 3

**3.1 a)** Considere que dispõe apenas do endereço de rede IP 130.XX.96.0/19, em que XX é o decimal correspondendo ao seu número de grupo (PLXX). 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. Assuma que todos os endereços de sub-redes são usáveis. Deve justificar as opções usadas .

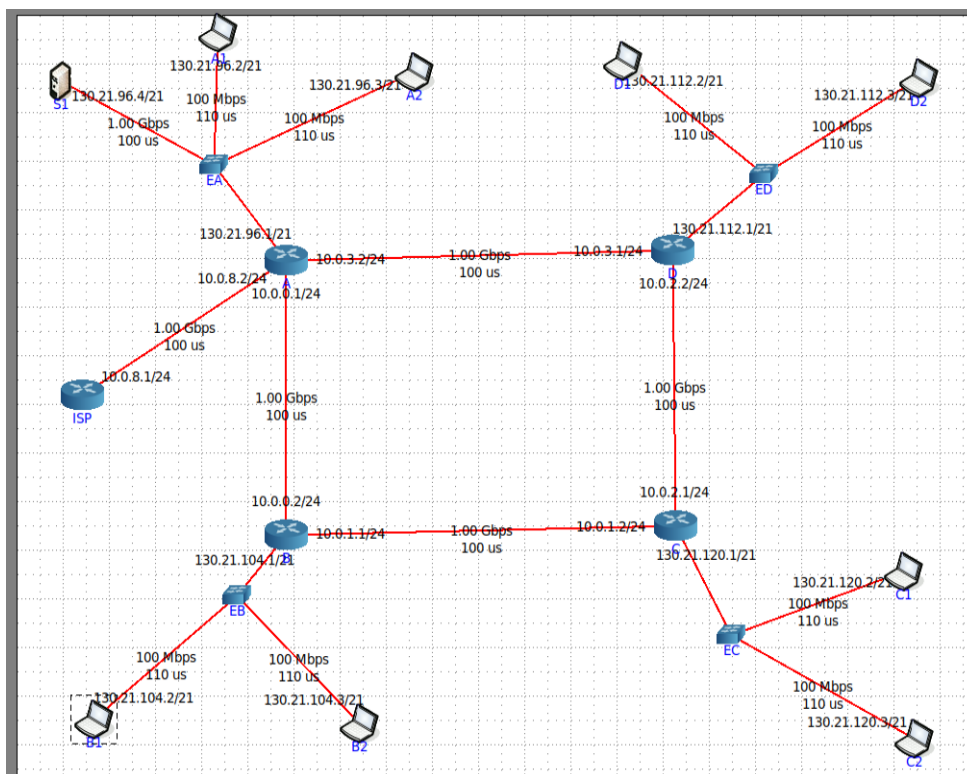


Figura 20: Topologia CORE após atribuição de novos IP

Para o novo esquema de endereçamento foi utilizado um endereço /21 onde os ultimos 2 bits do prefixo de rede indicam uma de 4 subredes. Cada uma das subredes corresponde no nosso modelo a um departamento.

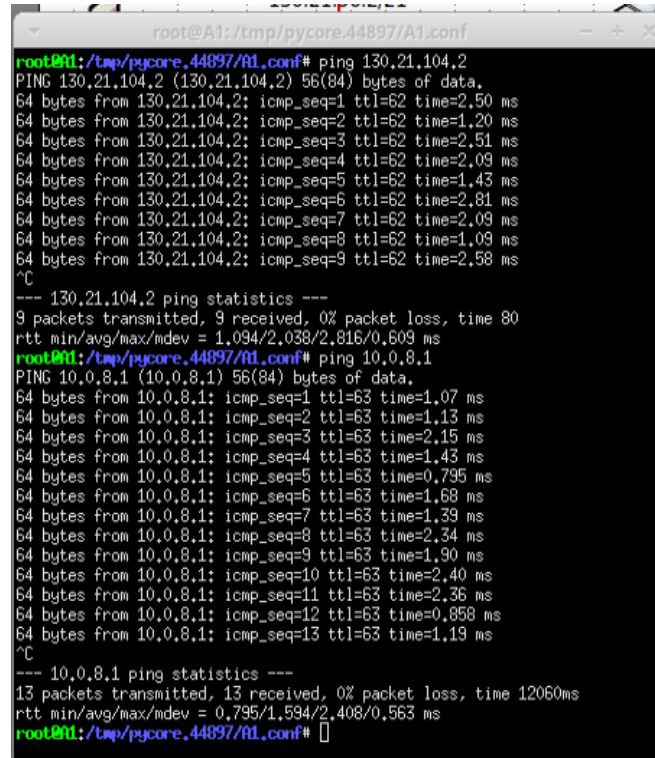
**3.2 b)** Qual a máscara de rede que usou (em formato decimal)? Quantos hosts IP pode interligar em cada departamento? Justifique.

A máscara de rede utilizada foi 255.255.248.0 porque o prefixo da rede passou a ter 21 bits, uma vez que, decidimos utilizar 2 bits para referenciar cada uma das quatro subredes. Como foram utilizados 21 bits para a máscara e o endereço tem no total 32,



sobram  $32-21=11$  bits para referenciar hosts dentro de cada departamento. Assim será possível interligar no máximo  $2^{11} = 2048$  hosts por cada departamento.

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



```
root@A1:/tmp/pycore.44897/A1.conf# ping 130.21.104.2
PING 130.21.104.2 (130.21.104.2) 56(84) bytes of data.
64 bytes from 130.21.104.2: icmp_seq=1 ttl=62 time=2.50 ms
64 bytes from 130.21.104.2: icmp_seq=2 ttl=62 time=1.20 ms
64 bytes from 130.21.104.2: icmp_seq=3 ttl=62 time=2.51 ms
64 bytes from 130.21.104.2: icmp_seq=4 ttl=62 time=2.09 ms
64 bytes from 130.21.104.2: icmp_seq=5 ttl=62 time=1.43 ms
64 bytes from 130.21.104.2: icmp_seq=6 ttl=62 time=2.81 ms
64 bytes from 130.21.104.2: icmp_seq=7 ttl=62 time=2.09 ms
64 bytes from 130.21.104.2: icmp_seq=8 ttl=62 time=1.09 ms
64 bytes from 130.21.104.2: icmp_seq=9 ttl=62 time=2.58 ms
^C
--- 130.21.104.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 80
rtt min/avg/max/mdev = 1.094/2.038/2.816/0.609 ms
root@A1:/tmp/pycore.44897/A1.conf# ping 10.0.8.1
PING 10.0.8.1 (10.0.8.1) 56(84) bytes of data.
64 bytes from 10.0.8.1: icmp_seq=1 ttl=63 time=1.07 ms
64 bytes from 10.0.8.1: icmp_seq=2 ttl=63 time=1.13 ms
64 bytes from 10.0.8.1: icmp_seq=3 ttl=63 time=2.15 ms
64 bytes from 10.0.8.1: icmp_seq=4 ttl=63 time=1.43 ms
64 bytes from 10.0.8.1: icmp_seq=5 ttl=63 time=0.795 ms
64 bytes from 10.0.8.1: icmp_seq=6 ttl=63 time=1.68 ms
64 bytes from 10.0.8.1: icmp_seq=7 ttl=63 time=1.33 ms
64 bytes from 10.0.8.1: icmp_seq=8 ttl=63 time=2.34 ms
64 bytes from 10.0.8.1: icmp_seq=9 ttl=63 time=1.90 ms
64 bytes from 10.0.8.1: icmp_seq=10 ttl=63 time=2.40 ms
64 bytes from 10.0.8.1: icmp_seq=11 ttl=63 time=2.36 ms
64 bytes from 10.0.8.1: icmp_seq=12 ttl=63 time=0.858 ms
64 bytes from 10.0.8.1: icmp_seq=13 ttl=63 time=1.19 ms
^C
--- 10.0.8.1 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12060ms
rtt min/avg/max/mdev = 0.795/1.594/2.408/0.563 ms
root@A1:/tmp/pycore.44897/A1.conf#
```

Figura 21: Conexão de departamento A para B e router ISP

```
root@B1: /tmp/pycore.44897/B1.conf
root@B1: /tmp/pycore.44897/B1.conf# ping 130.21.120.2
PING 130.21.120.2 (130.21.120.2) 56(84) bytes of data.
64 bytes from 130.21.120.2: icmp_seq=1 ttl=62 time=1.05 ms
64 bytes from 130.21.120.2: icmp_seq=2 ttl=62 time=2.14 ms
64 bytes from 130.21.120.2: icmp_seq=3 ttl=62 time=0.890 ms
64 bytes from 130.21.120.2: icmp_seq=4 ttl=62 time=1.91 ms
64 bytes from 130.21.120.2: icmp_seq=5 ttl=62 time=2.09 ms
64 bytes from 130.21.120.2: icmp_seq=6 ttl=62 time=1.82 ms
64 bytes from 130.21.120.2: icmp_seq=7 ttl=62 time=3.25 ms
64 bytes from 130.21.120.2: icmp_seq=8 ttl=62 time=1.80 ms
64 bytes from 130.21.120.2: icmp_seq=9 ttl=62 time=1.27 ms
64 bytes from 130.21.120.2: icmp_seq=10 ttl=62 time=3.18 ms
64 bytes from 130.21.120.2: icmp_seq=11 ttl=62 time=2.91 ms
^C
--- 130.21.120.2 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 10027ms
rtt min/avg/max/mdev = 0.890/2.031/3.251/0.772 ms
root@B1: /tmp/pycore.44897/B1.conf# ping 10.0.8.1
PING 10.0.8.1 (10.0.8.1) 56(84) bytes of data.
64 bytes from 10.0.8.1: icmp_seq=1 ttl=62 time=3.20 ms
64 bytes from 10.0.8.1: icmp_seq=2 ttl=62 time=2.95 ms
64 bytes from 10.0.8.1: icmp_seq=3 ttl=62 time=4.38 ms
64 bytes from 10.0.8.1: icmp_seq=4 ttl=62 time=4.04 ms
64 bytes from 10.0.8.1: icmp_seq=5 ttl=62 time=2.40 ms
64 bytes from 10.0.8.1: icmp_seq=6 ttl=62 time=1.36 ms
64 bytes from 10.0.8.1: icmp_seq=7 ttl=62 time=2.39 ms
64 bytes from 10.0.8.1: icmp_seq=8 ttl=62 time=2.34 ms
^C
--- 10.0.8.1 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 7009ms
rtt min/avg/max/mdev = 1.369/2.888/4.384/0.918 ms
root@B1: /tmp/pycore.44897/B1.conf#
```

Figura 22: Conexão de departamento B para C e router ISP

```
root@C:/tmp/pycore.44897/C.conf# ping 130.21.112.2
PING 130.21.112.2 (130.21.112.2) 56(84) bytes of data,
64 bytes from 130.21.112.2: icmp_seq=1 ttl=63 time=2.03 ms
64 bytes from 130.21.112.2: icmp_seq=2 ttl=63 time=2.42 ms
64 bytes from 130.21.112.2: icmp_seq=3 ttl=63 time=1.44 ms
64 bytes from 130.21.112.2: icmp_seq=4 ttl=63 time=1.57 ms
64 bytes from 130.21.112.2: icmp_seq=5 ttl=63 time=2.25 ms
64 bytes from 130.21.112.2: icmp_seq=6 ttl=63 time=2.08 ms
64 bytes from 130.21.112.2: icmp_seq=7 ttl=63 time=1.72 ms
64 bytes from 130.21.112.2: icmp_seq=8 ttl=63 time=2.35 ms
64 bytes from 130.21.112.2: icmp_seq=9 ttl=63 time=2.26 ms
^C
--- 130.21.112.2 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8013ms
rtt min/avg/max/mdev = 1.440/2.018/2.421/0.337 ms
root@C:/tmp/pycore.44897/C.conf# ping 10.0.8.1
PING 10.0.8.1 (10.0.8.1) 56(84) bytes of data,
64 bytes from 10.0.8.1: icmp_seq=1 ttl=62 time=2.31 ms
64 bytes from 10.0.8.1: icmp_seq=2 ttl=62 time=3.05 ms
64 bytes from 10.0.8.1: icmp_seq=3 ttl=62 time=3.31 ms
64 bytes from 10.0.8.1: icmp_seq=4 ttl=62 time=2.34 ms
64 bytes from 10.0.8.1: icmp_seq=5 ttl=62 time=2.96 ms
64 bytes from 10.0.8.1: icmp_seq=6 ttl=62 time=2.06 ms
64 bytes from 10.0.8.1: icmp_seq=7 ttl=62 time=2.25 ms
64 bytes from 10.0.8.1: icmp_seq=8 ttl=62 time=3.62 ms
64 bytes from 10.0.8.1: icmp_seq=9 ttl=62 time=2.53 ms
64 bytes from 10.0.8.1: icmp_seq=10 ttl=62 time=3.79 ms
64 bytes from 10.0.8.1: icmp_seq=11 ttl=62 time=2.83 ms
64 bytes from 10.0.8.1: icmp_seq=12 ttl=62 time=4.07 ms
64 bytes from 10.0.8.1: icmp_seq=13 ttl=62 time=3.35 ms
^C
--- 10.0.8.1 ping statistics ---
13 packets transmitted, 13 received, 0% packet loss, time 12016ms
rtt min/avg/max/mdev = 2.069/2.964/4.077/0.621 ms
root@C:/tmp/pycore.44897/C.conf#
```

Figura 23: Conexão de departamento C para D e router ISP

```
root@D1:/tmp/pycore.44897/D1.conf# ping 10.0.8.1
PING 10.0.8.1 (10.0.8.1) 56(84) bytes of data,
64 bytes from 10.0.8.1: icmp_seq=1 ttl=62 time=2.84 ms
64 bytes from 10.0.8.1: icmp_seq=2 ttl=62 time=1.43 ms
64 bytes from 10.0.8.1: icmp_seq=3 ttl=62 time=3.05 ms
64 bytes from 10.0.8.1: icmp_seq=4 ttl=62 time=2.60 ms
64 bytes from 10.0.8.1: icmp_seq=5 ttl=62 time=2.64 ms
64 bytes from 10.0.8.1: icmp_seq=6 ttl=62 time=1.21 ms
64 bytes from 10.0.8.1: icmp_seq=7 ttl=62 time=1.88 ms
64 bytes from 10.0.8.1: icmp_seq=8 ttl=62 time=1.33 ms
64 bytes from 10.0.8.1: icmp_seq=9 ttl=62 time=1.20 ms
64 bytes from 10.0.8.1: icmp_seq=10 ttl=62 time=3.95 ms
^C
--- 10.0.8.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9014ms
rtt min/avg/max/mdev = 1.207/2.217/3.957/0.894 ms
root@D1:/tmp/pycore.44897/D1.conf#
```

Figura 24: Conexão entre o departamento D e o router ISP

Para verificar a conectividade entre todos os departamentos foi testada a conexão entre os varios departamentos entre si e de cada um para o router ISD que não pertence a nenhum departamento. Como se pode verificar pelas figuras acima, existe conexão entre os departamentos A e B, B e C, C e D e de cada um para o router ISP, garantindo assim a conectividade IP entre as várias redes locais da organização MIEI-RC.

## Conclusão

Este trabalho encontra-se dividido em duas partes. A primeira parte foca-se no registo de datagramas IP, e no processo de fragmentação, enquanto que a segunda aborda de uma forma mais aprofundada o endereçamento e encaminhamento IP, dando continuidade ao estudo do protocolo IPv4.

Primeiramente, com o objetivo de estudar o *Internet Protocol* (IP) foi construída uma Topologia CORE, através de uma máquina virtual previamente disponibilizada, de forma a permitir a simulação de redes. De forma a capturar e analisar o tráfego IP utilizamos a ferramenta *Wireshark*, e para tal usamos o programa *traceroute* para descobrir uma rota IP, enviando pacotes de diferentes tamanhos para o seu destino. E, desta forma, foi conseguido verificar qual o valor inicial mínimo do campo TTL de forma a alcançar um servidor.

Na segunda parte, construímos novamente uma Topologia CORE, e observamos as máscaras de rede que foram atribuídas a cada equipamento. Além disso, analisamos tabelas de encaminhamento, abordamos conceitos como o encaminhamento estático ou dinâmico e também trabalhamos com rotas estáticas.

Resumidamente, a compreensão de conceitos relacionados com o capítulo do Protocolo IP, tal como por exemplo o *sub-netting*, foi clarificada e aprofundada com a realização deste trabalho, conjuntamente com os conhecimentos obtidos ao longo das aulas teóricas.