

Universidade do Minho



# Redes de Computadores

## RC-TP2

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

3º ANO

GRUPO 61

RENATO ANDRÉ ARAÚJO AZEVEDO  
GONÇALO COSTA DE ALMEIDA  
MARIA SOFIA MARTINHO GONÇALVES JORDÃO MARQUES

**A89547**  
**A88292**  
**A87963**

# Índice

<b>1</b>	<b>Introdução</b>	<b>3</b>
<b>2</b>	<b>Parte 1</b>	<b>3</b>
2.1	Ex1 . . . . .	3
2.1.1	Alínea a . . . . .	3
2.1.2	Alínea b . . . . .	4
2.1.3	Alínea c . . . . .	4
2.1.4	Alínea d . . . . .	4
2.2	Ex2 . . . . .	5
2.2.1	Alinea a . . . . .	5
2.2.2	Alinea b . . . . .	5
2.2.3	Alinea c . . . . .	5
2.2.4	Alinea d . . . . .	6
2.2.5	Alinea e . . . . .	6
2.2.6	Alinea f . . . . .	7
2.2.7	Alinea g . . . . .	7
2.3	Ex3 . . . . .	8
2.3.1	Alinea a . . . . .	8
2.3.2	Alinea b . . . . .	9
2.3.3	Alinea c . . . . .	9
2.3.4	Alinea d . . . . .	10
2.3.5	Alinea e . . . . .	10
<b>3</b>	<b>Parte 2</b>	<b>11</b>
3.1	Ex1 . . . . .	11
3.1.1	Alinea a . . . . .	11
3.1.2	Alinea b . . . . .	11
3.1.3	Alinea c . . . . .	12
3.1.4	Alinea d . . . . .	12
3.1.5	Alinea e . . . . .	13
3.2	Ex2 . . . . .	14
3.2.1	Alinea a . . . . .	14
3.2.2	Alinea b . . . . .	15
3.2.3	Alinea c . . . . .	16
3.2.4	Alinea d . . . . .	16
3.2.5	Alinea e . . . . .	16
3.3	Ex3 . . . . .	17
3.3.1	Alinea 1 . . . . .	18
3.3.2	Alinea 2 . . . . .	18
3.3.3	Alinea 3 . . . . .	18
<b>4</b>	<b>Conclusão</b>	<b>19</b>

# 1 Introdução

No âmbito da cadeira de Redes Computadores foi-nos proposta a elaboração de um trabalho composto por duas partes que terá como principal objetivo o estudo do Internet Protocol (IP) nas suas principais vertentes, nomeadamente: estudo do formato de um pacote ou datagrama IP, fragmentação de pacotes IP, endereçamento IP e encaminhamento IP.

Na primeira parte deste projeto será realizado o registo de datagramas IP estes serão analisados os vários campos de um datagrama IP e detalhado o processo de fragmentação realizado pelo IP. Na segunda parte deste projeto continua-se o estudo do protocolo IPv4 com ênfase no endereçamento e encaminhamento IP. Serão também apresentadas algumas das técnicas mais relevantes que foram propostas para aumentar a escalabilidade do protocolo IP, apaziguar a exaustão dos endereços IPv4 e também reduzir os recursos de memória necessários nos routers para manter as tabelas de encaminhamento.

Esperamos, com este trabalho, atingir os objetivos definidos pelo docente.

## 2 Parte 1

### 2.1 Ex1

Prepare uma topologia CORE para verificar o comportamento do traceroute. Ligue um host (pc) Cliente1 a um router R2; o router R2 a um router R3, que por sua vez, se liga a um host (servidor) Servidor1. (Note que pode não existir conectividade IP imediata entre o Cliente1 e o Servidor1 até que o anúncio de rotas estabilize). Ajuste o nome dos equipamentos atribuídos por defeito para a topologia do enunciado.

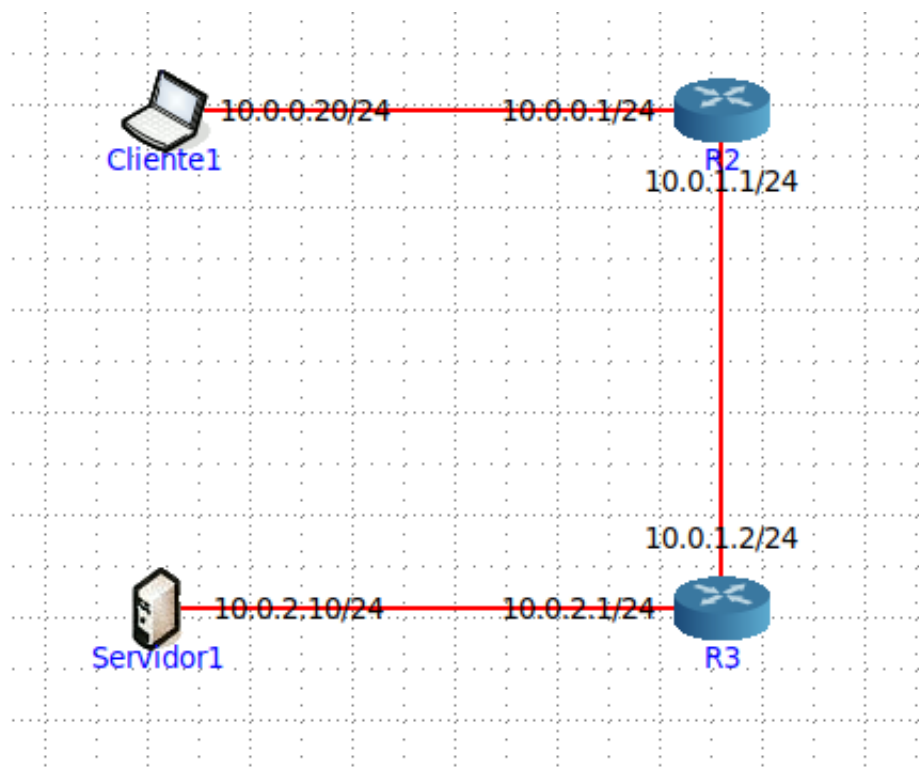


Figure 1: Topologia Core

#### 2.1.1 Alínea 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.44181/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)  0.084 ms  0.009 ms  0.006 ms
 2  10.0.1.2 (10.0.1.2)  0.019 ms  0.009 ms  0.009 ms
 3  10.0.2.10 (10.0.2.10)  0.031 ms  0.012 ms  0.012 ms
root@Cliente1:/tmp/pycore.44181/Cliente1.conf#

```

Figure 2: Shell com o traceroute -I

### 2.1.2 Alínea b

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

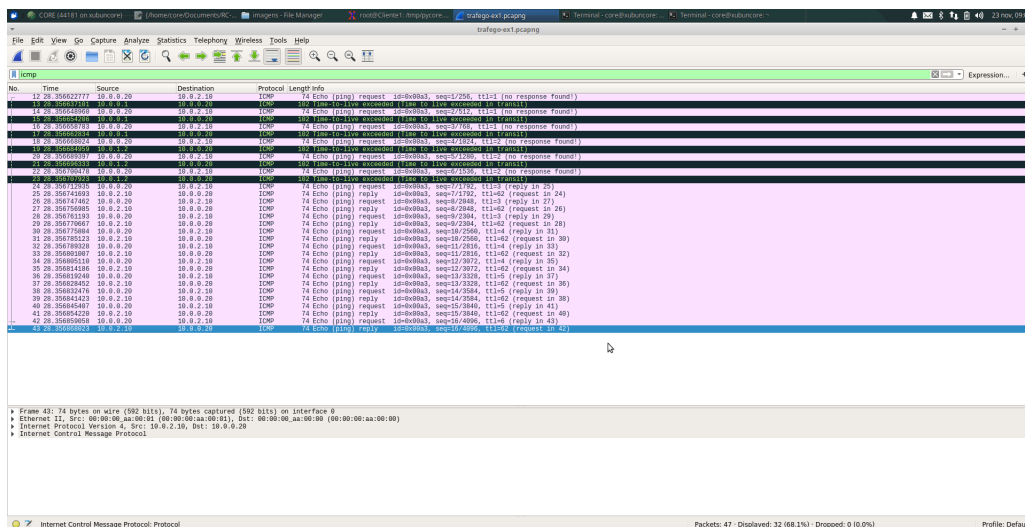


Figure 3: Trafego do wireshark

Em cada salto que a mensagem efetua, o TTL é decrementado em 1. Desta forma será de esperar que enquanto o TTL não for 3, ou seja, igual ao número de saltos, vamos receber respostas TTL exceeded. Na prática, o cliente envia 3 pings com TTL = 1 para o R1, onde o R1 responde com TTL exceeded, tal como seria de esperar, uma vez que o TTL foi decrementado pelo router R1, chegando a 0. Depois o mesmo acontece para o R2, uma vez que o router R1 reduziu o TTL para 1. Por fim, quando TTL = 3, a mensagem consegue chegar ao Servidor1, obtendo uma resposta.

### 2.1.3 Alínea c

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

Para alcançar o servidor1, o TTL esperado seria de 3, uma vez que se encontra a 3 saltos do cliente1. O que se verifica na prática, tal como se observa na imagem anterior, onde deixamos de receber TTL exceeded quando o TTL chega a 3.

### 2.1.4 Alínea d

Calcule o valor médio do tempo de ida-e-volta (Round-TripTime) obtido?

Os tempos obtidos foram:

- Para o R2: 0.084 ms, 0.009 ms, 0.006 ms, com uma média de 0.033 ms
- Para o R3: 0.019 ms, 0.009 ms, 0.009 ms, com uma média de 0.012 ms
- Para o Servidor1: 0.031 ms, 0.012 ms, 0.012 ms, com uma média de 0.018 ms

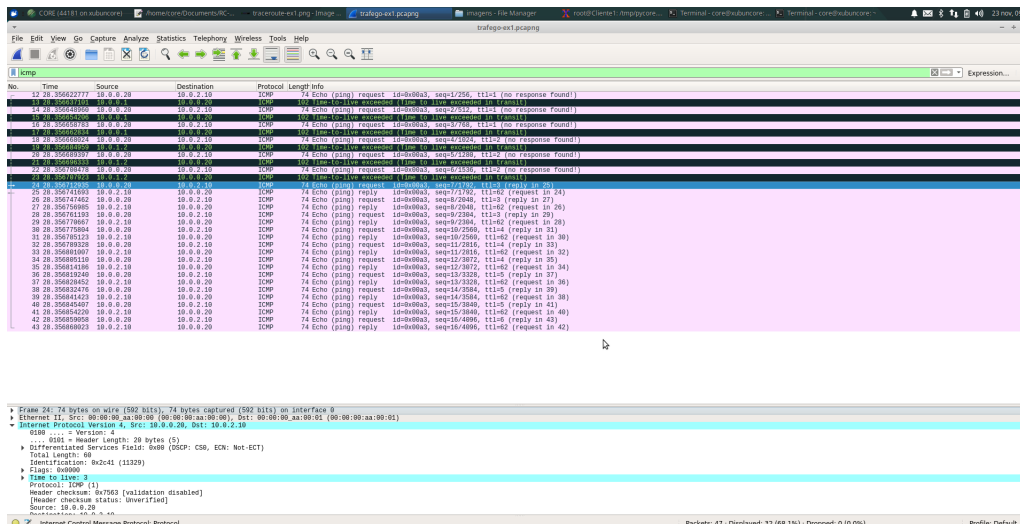


Figure 4: TTL ex 1

## 2.2 Ex2

Selecione a primeira mensagem ICMP capturada (referente a (i) tamanho por defeito) e centre a análise no nível protocolar IP (expanda o tab correspondente na janela de detalhe do wireshark). Através da análise do cabeçalho IP diga:

### 2.2.1 Alinea a

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

O valor do IP da interface do computador é 172.26.47.154.

### 2.2.2 Alinea b

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

Como podemos observar na imagem, o campo do protocolo tem valor 1, o que corresponde ao protocolo ICMP.

```
> Frame 3: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: IntelCor_25:3e:15 (18:56:00:25:3e:15), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.47.154, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x23b3 (9139)
  > Flags: 0x00
    Fragment Offset: 0
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.47.154
    Destination Address: 193.136.9.240
> Internet Control Message Protocol
```

Figure 5: Valor do campo protocolo

### 2.2.3 Alinea 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) possui 20 bytes, pelo *Header Length*. Como o *Total Length* possui 56 bytes, então o payload deverá ter 36 bytes, uma vez que o  $Total\ Length = Header\ Length + payload$ .

```
> Frame 3: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: IntelCor_25:3e:15 (18:56:80:25:3e:15), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.47.154, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x23b3 (9139)
  > Flags: 0x00
    Fragment Offset: 0
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.47.154
    Destination Address: 193.136.9.240
> Internet Control Message Protocol
```

Figure 6: Calculo do payload

#### 2.2.4 Alinea d

**O datagrama IP foi fragmentado? Justifique.**

Como  $Flags = 0s$ ,  $Fragment\ Offset = 0$ , podemos concluir que a mensagem não foi fragmentada, uma vez que o *Fragment Offset* a zero indica que seria o primeiro fragmento, e a flag *More fragments* a zero indica que não existem mais fragmentos.

```
Flags: 0x00
  0... .... = Reserved bit: Not set
  .0.. .... = Don't fragment: Not set
  ..0. .... = More fragments: Not set
Fragment Offset: 0
```

Figure 7: Flags

#### 2.2.5 Alinea 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 valores que se alteram são o identification, e o *time to live*.

```
> Frame 5: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: IntelCor_25:3e:15 (18:56:80:25:3e:15), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.47.154, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x23b4 (9140)
  > Flags: 0x00
    Fragment Offset: 0
  > Time to Live: 1
  > Protocol: ICMP (1)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.47.154
    Destination Address: 193.136.9.240
> Internet Control Message Protocol
```

Figure 8: Variação dos valores 1

```

> Frame 6: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: IntelCor_25:3e:15 (18:56:80:25:3e:15), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
  > Internet Protocol Version 4, Src: 172.26.47.154, Dst: 193.136.9.240
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x23b5 (9141)
    Flags: 0x00
    Fragment Offset: 0
  > Time to Live: 2
    Protocol: ICMP (1)
    Header Checksum: 0x0000 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.47.154
    Destination Address: 193.136.9.240
  > Internet Control Message Protocol

```

Figure 9: Variação dos valores 2

## 2.2.6 Alinea f

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

Em cada datagrama temos que  $identification_{i+1} = 1 + identification_i$ .

Em cada datagrama temos que o valor de  $TTL_{i+1}$  será:

- 1 se  $TTL_i = 255$
- 255 se  $TTL_i = 3$
- $TTL_i + 1$  caso contrário

3	0.801745	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5539/41749, ttl=255 (reply in 4)
5	0.840366	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5540/42005, ttl=1 (no response found!)
6	0.879438	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5541/42261, ttl=2 (no response found!)
21	0.917501	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5542/42517, ttl=3 (no response found!)
23	0.956808	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5543/42773, ttl=4 (reply in 24)
37	3.302767	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5544/43029, ttl=255 (reply in 39)
38	3.341497	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5545/43285, ttl=1 (no response found!)
41	3.380380	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5546/43541, ttl=2 (no response found!)
43	3.418585	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5547/43797, ttl=3 (no response found!)
45	3.456670	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5548/44053, ttl=4 (reply in 46)
54	5.803058	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5549/44309, ttl=255 (reply in 55)
56	5.853767	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5550/44565, ttl=1 (no response found!)
58	5.903723	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5551/44821, ttl=2 (no response found!)
60	5.954229	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5552/45077, ttl=3 (no response found!)
62	6.005098	172.26.47.154	193.136.9.240	ICMP	70 Echo (ping) request	id=0x0001, seq=5553/45333, ttl=4 (reply in 63)

Figure 10: Valores do TTL

## 2.2.7 Alinea 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?

Nas primeiras respostas ICMP TTL exceeded temos um valor TTL inicial de 255 que vai diminuindo até 253. Isto acontece porque a mensagem faz 3 saltos até ao destino, onde o TTL é decrementado em 1 por cada salto. Na primeira mensagem recebemos o TTL máximo de 255, uma vez que a mensagem só chegou até ao primeiro router, não sendo decrementado por nenhum router. Depois, na segunda mensagem, o TTL será 254, por ter sido decrementado em 1 pelo router intermédio. Por fim será de 253, uma vez que passou por dois routers intermédios, tendo sido decrementado duas vezes.



```

> Frame 57: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: IntelCor_25:3e:15 (18:56:80:25:3e:15)
▼ Internet Protocol Version 4, Src: 172.26.254.254, Dst: 172.26.47.154
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
        Total Length: 56
        Identification: 0xeed (61165)
    > Flags: 0x00
        Fragment Offset: 0
        Time to Live: 255
        Protocol: ICMP (1)
        Header Checksum: 0x4549 [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 172.26.254.254
        Destination Address: 172.26.47.154
    > Internet Control Message Protocol

```

Figure 11: Primeiro valor TTL

```

> Frame 59: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: IntelCor_25:3e:15 (18:56:80:25:3e:15)
▼ Internet Protocol Version 4, Src: 172.16.2.1, Dst: 172.26.47.154
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 56
        Identification: 0x77eb (30699)
    > Flags: 0x00
        Fragment Offset: 0
        Time to Live: 254
        Protocol: ICMP (1)
        Header Checksum: 0xbb13 [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 172.16.2.1
        Destination Address: 172.26.47.154
    > Internet Control Message Protocol

```

Figure 12: Segundo valor TTL

```

> Frame 61: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: IntelCor_25:3e:15 (18:56:80:25:3e:15)
▼ Internet Protocol Version 4, Src: 172.16.115.252, Dst: 172.26.47.154
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
        Total Length: 56
        Identification: 0xefba (61370)
    > Flags: 0x00
        Fragment Offset: 0
        Time to Live: 253
        Protocol: ICMP (1)
        Header Checksum: 0xd248 [validation disabled]
        [Header checksum status: Unverified]
        Source Address: 172.16.115.252
        Destination Address: 172.26.47.154
    > Internet Control Message Protocol

```

Figure 13: Terceiro valor TTL

## 2.3 Ex3

Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempode captura. Observe o tráfego depois do tamanho de pacote ter sido definido para 32XX bytes.

### 2.3.1 Alinea a

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

A mensagem teve de ser fragmentada, uma vez que a quantidade de bytes que se pretende enviar é superior ao máximo definido, sendo fragmentada em 2 mensagens com 1514 bytes e 1 mensagem de 315 bytes.



47	50.854186	172.26.47.154	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=23c3) [Reassembled in #49]
48	50.854186	172.26.47.154	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=23c3) [Reassembled in #49]
49	50.854186	172.26.47.154	193.136.9.240	ICMP	315	Echo (ping) request id=0x0001, seq=5555/45845, ttl=1 (no response found!)

Figure 14: Mensagem fragmentada

### 2.3.2 Alinea 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* permite verificar se o texto foi fragmentado. Como esta como Set, podemos confirmar a existencia de mais fragmentos. Para saber que se trata do primeiro fragmento, basta olhar para o campo *Fragment Offset*, e como este está a 0, podemos saber que este é o primeiro fragmento. Como *Total Length* = 1500, sabemos que o tamanho da mensagem é de 1500 bytes, sendo 20 para o cabeçalho e 1480 para o payload).

```
> Frame 47: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: IntelCor_25:3e:15 (18:56:80:25:3e:15), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.47.154, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x23c3 (9155)
> Flags: 0x20, More fragments
  0... .... = Reserved bit: Not set
  .0... .... = Don't fragment: Not set
  ..1. .... = More fragments: Set
  Fragment Offset: 0
> Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.47.154
  Destination Address: 193.136.9.240
  [Reassembled IPv4 in frame: 49]
> Data (1480 bytes)
```

Figure 15: Primeiro fragmento do datagrama IP

### 2.3.3 Alinea 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?

Como o *Fragment Offset* = 1480, podemos concluir que esta se trata da segunda mensagem, já que a primeira tem um *Total Length* de 1500, onde 1480 são para o *payload*, sendo esse o *Offset* da segunda mensagem.

```
> Frame 48: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: IntelCor_25:3e:15 (18:56:80:25:3e:15), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.47.154, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1500
  Identification: 0x23c3 (9155)
> Flags: 0x20, More fragments
  0... .... = Reserved bit: Not set
  .0... .... = Don't fragment: Not set
  ..1. .... = More fragments: Set
  Fragment Offset: 1480
> Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.47.154
  Destination Address: 193.136.9.240
  [Reassembled IPv4 in frame: 49]
> Data (1480 bytes)
```

Figure 16: Segundo fragmento do datagrama IP

### 2.3.4 Alinea d

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

Foram criados 3 fragmentos, podendo identificar o último através da flag *More fragments*, que está Not set, o que significa que não existem mais fragmentos. Sendo assim podemos identificar 3 fragmentos da mensagem, como vimos nas imagens anteriores.

```
> Frame 49: 315 bytes on wire (2520 bits), 315 bytes captured (2520 bits) on interface \Device\NPF_{9C19C47F-9AEF-4A87-BB88-34E08FD1F168}, id 0
> Ethernet II, Src: IntelCor_25:3e:15 (18:56:80:25:3e:15), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
> Internet Protocol Version 4, Src: 172.26.47.154, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 301
  Identification: 0x23c3 (9155)
  Flags: 0x01
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  Fragment Offset: 2960
> Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0x0000 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.47.154
  Destination Address: 193.136.9.240
  [3 IPv4 Fragments (3241 bytes): #47(1480), #48(1480), #49(281)]
> Internet Control Message Protocol
```

Figure 17: Ultimo fragmento do datagrama IP

### 2.3.5 Alinea 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 se alteram no cabeçalho IP dos 3 fragmentos são *More fragments* e o *Fragment Offset*. Para reconstruir o datagrama original, devemos organizar os fragmentos por ordem crescente do offset, tendo em consideração que o ultimo fragmento será o que tiver a flag *More fragments* a zero, visto que esta indica que não existem mais fragmentos desse datagrama.



### 3.1.3 Alinea c

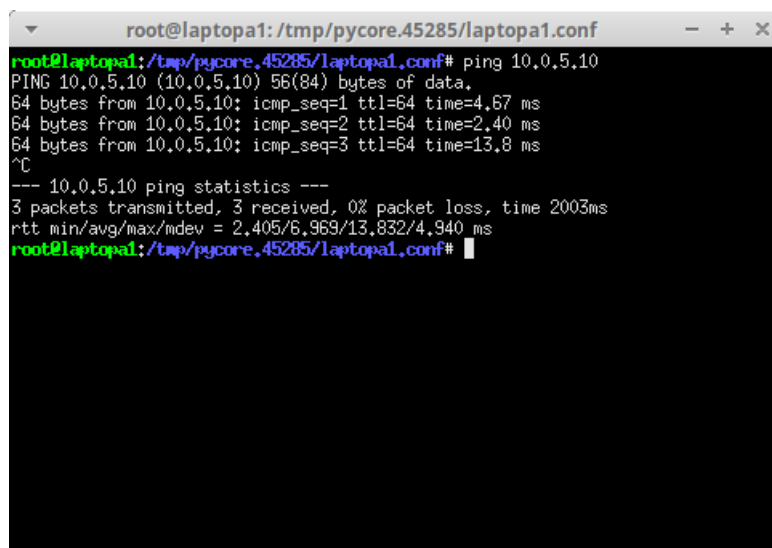
Porque razão não é atribuído um endereço IP aos switches?

Os switches não tem ip uma vez que apenas trabalham na camada physical e link, ao contrário dos routers que por sua vez já trabalham na camada do network, daí serem lhes atribuído ips.

### 3.1.4 Alinea d

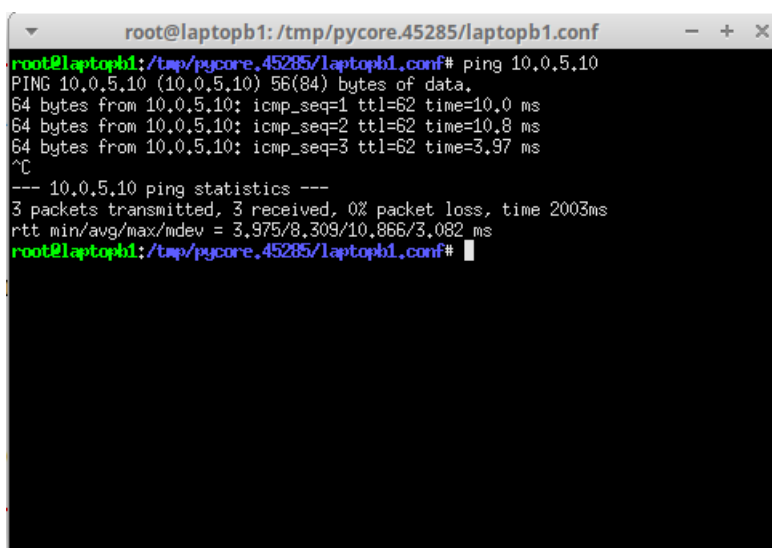
Usando o comando ping certifique-se que existe conectividade IP entre os laptops dos vários departamentos e o servidor do departamento A (basta certificar-se da conectividade de um laptop por departamento).

Como mostram as imagens, podemos concluir que existe conexão entre os computadores dos departamentos e o servidor S<sub>1</sub>.



```
root@laptopa1: /tmp/pycore.45285/laptopa1.conf
root@laptopa1:/tmp/pycore.45285/laptopa1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_seq=1 ttl=64 time=4.67 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=2.40 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=64 time=13.8 ms
^C
--- 10.0.5.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 2.405/6.969/13.832/4.940 ms
root@laptopa1:/tmp/pycore.45285/laptopa1.conf#
```

Figure 19: Conctividade no departamento A



```
root@laptopb1: /tmp/pycore.45285/laptopb1.conf
root@laptopb1:/tmp/pycore.45285/laptopb1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=10.0 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=10.8 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=3.97 ms
^C
--- 10.0.5.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 3.975/8.309/10.866/3.082 ms
root@laptopb1:/tmp/pycore.45285/laptopb1.conf#
```

Figure 20: Conctividade no departamento B

```
root@laptopc1: /tmp/pycore.45285/laptopc1.conf
root@laptopc1: /tmp/pycore.45285/laptopc1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=61 time=11.8 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=28.9 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=17.3 ms
^C
--- 10.0.5.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 11.892/19.418/28.991/7.130 ms
root@laptopc1: /tmp/pycore.45285/laptopc1.conf#
```

Figure 21: Conctividade no departamento C

```
root@laptopd1: /tmp/pycore.45285/laptopd1.conf
root@laptopd1: /tmp/pycore.45285/laptopd1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=9.77 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=3.87 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=23.7 ms
^C
--- 10.0.5.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 3.877/12.451/23.702/8.311 ms
root@laptopd1: /tmp/pycore.45285/laptopd1.conf#
```

Figure 22: Conctividade no departamento D

### 3.1.5 Alinea e

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

Como podemos ver na imagem, existe conexão entre o router de acesso  $R_{isp}$  e o servidor  $S_1$ .

```
root@Risp: /tmp/pycore.45285/Risp.conf
root@Risp:/tmp/pycore.45285/Risp.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_seq=1 ttl=63 time=1.31 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=63 time=1.54 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=63 time=13.6 ms
^C
--- 10.0.5.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2006ms
rtt min/avg/max/mdev = 1.311/5.516/13.695/5.784 ms
root@Risp:/tmp/pycore.45285/Risp.conf#
```

Figure 23: Conctividade no router ISP

## 3.2 Ex2

Para o router e um laptop do departamento C:

### 3.2.1 Alinea 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; interpreteas várias entradas de cada tabela. Se necessário, consulte o manual respetivo (`man netstat`).

Nas tabelas de encaminhamento, a coluna *Destination* representa o IP do destino da mensagem. Já a coluna *Gateway* representa o IP do proximo nodo onde será enviada a mensagem. Por fim, a coluna *Genmask* representa a máscara aplicada ao IP destino.

```
root@laptopc1: /tmp/pycore.34411/laptopc1.conf
root@laptopc1:/tmp/pycore.34411/laptopc1.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@laptopc1:/tmp/pycore.34411/laptopc1.conf#
```

Figure 24: Tabela de encaminhamento de um laptop

A primeira entrada na tabela representa a rota por defeito, encaminhando a mensagem para o router C. Já a segunda entrada, representa a rota com destino para a rede local.

```

root@Rc: /tmp/pycore.34411/Rc.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0          10.0.3.2        255.255.255.0   UG        0 0          0 eth1
10.0.1.0          10.0.2.1        255.255.255.0   UG        0 0          0 eth0
10.0.2.0          0.0.0.0         255.255.255.0   U         0 0          0 eth0
10.0.3.0          0.0.0.0         255.255.255.0   U         0 0          0 eth1
10.0.4.0          10.0.2.1        255.255.255.0   UG        0 0          0 eth0
10.0.5.0          10.0.2.1        255.255.255.0   UG        0 0          0 eth0
10.0.6.0          10.0.3.2        255.255.255.0   UG        0 0          0 eth1
10.0.7.0          0.0.0.0         255.255.255.0   U         0 0          0 eth2
10.0.8.0          10.0.2.1        255.255.255.0   UG        0 0          0 eth0
root@Rc: /tmp/pycore.34411/Rc.conf#

```

Figure 25: Tabela de encaminhamento do router C

Nesta tabela de encaminhamento, podemos observar entradas para todas as subredes da rede definida, e o nodo destino. Por exemplo, a primeira entrada da tabela representa o caminho para a subrede representada pelo IP 10.0.0.0, que corresponde a subrede entre os router A e D, onde o proximo nodo é o 10.0.3.2. Em todas as entradas da tabela, a máscara é 255.255.255.0.

### 3.2.2 Alinea 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`).

O router  $R_c$  está a usar encaminhamento dinâmico, como podemos observar pela existencia do processo `ospf`, que é um processo de routing dinâmico, enquanto que o laptop da rede não possui nenhum processo relacionado a IPs dinâmicos.

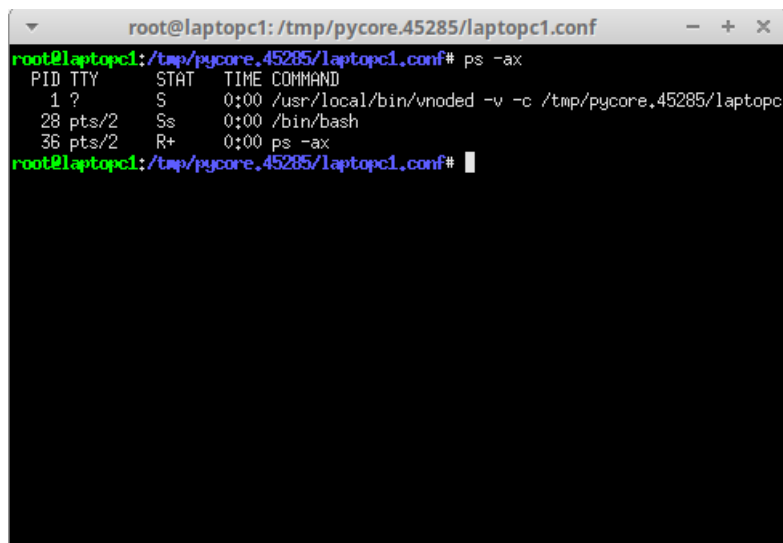
```

root@Rc: /tmp/pycore.45285/Rc.conf# ps -ax
PID TTY      STAT   TIME COMMAND
  1 ?        S      0:00 /usr/local/bin/vnoded -v -c /tmp/pycore.45285/Rc -l /
 59 ?        Ss     0:00 /usr/sbin/zebra -d
 65 ?        Ss     0:00 /usr/sbin/ospfd -d
 69 ?        Ss     0:00 /usr/sbin/ospfd -d
 77 pts/2    Ss     0:00 /bin/bash
 85 pts/2    R+     0:00 ps -ax
root@Rc: /tmp/pycore.45285/Rc.conf#

```

Figure 26: Processos do router





```
root@laptopc1: /tmp/pycore.45285/laptopc1.conf# ps -ax
PID TTY      STAT   TIME COMMAND
  1  ?        S      0:00 /usr/local/bin/vnoded -v -c /tmp/pycore.45285/laptopc
 28 pts/2    Ss     0:00 /bin/bash
 36 pts/2    R+     0:00 ps -ax
root@laptopc1: /tmp/pycore.45285/laptopc1.conf#
```

Figure 27: Processos de um laptop

### 3.2.3 Alinea 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.

Ao remover a rota por defeito o servidor  $S_1$  deixa de conseguir responder a quem não se encontra na sua subrede, apesar de conseguirem aceder ao servidor. Isto acontece, porque apesar de todos os elementos da rede conhecerem o servidor  $S_1$ , este apenas consegue encaminhar mensagens para a sua subrede.

### 3.2.4 Alinea 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.

Para restabelecer a conectividade para o servidor  $S_1$ , é necessário adicionar à tabela de encaminhamento do servidor rotas para as três subredes (10.6.0.0, 10.0.7.0, e 10.0.8.0).

Os comandos utilizados para restabelecer as rotas foram:

- `route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.5.1`
- `route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.5.1`
- `route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.5.1`

### 3.2.5 Alinea 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.

Depois de adicionar as novas entradas à tabela de encaminhamento, a conexão foi restabelecida, como podemos ver na imagem.

```
root@laptopb2: /tmp/pycore.45285/laptopb2.conf
root@laptopb2:/tmp/pycore.45285/laptopb2.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_seq=1 ttl=62 time=7.95 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=62 time=4.40 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=62 time=13.3 ms
^C
--- 10.0.5.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 4.408/8.585/13.397/3.698 ms
root@laptopb2:/tmp/pycore.45285/laptopb2.conf#
```

Figure 28: Teste de conectividade

### 3.3 Ex3

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. Recordar-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.

### 3.3.1 Alinea 1

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.

Na nossa resolução decidimos usar os 5 bits do terceiro octeto para identificar as nossas subredes e os restantes 8 bits para atribuição de ips. Esta escolha foi feita de forma a permitir crescimento tanto ao nível de subredes como na distribuição de ips, uma vez que no modelo atual, seriam necessários apenas 2 bits para representar as 4 subredes e 3 bits para representar as diferentes interfaces em cada subrede.

Desta forma, atribuímos os endereços 130.61.96.0 / 24 ao departamento A, 130.61.97.0 / 24 ao departamento B, 130.61.98.0 / 24 ao departamento C, e por fim, 130.61.99.0 / 24 ao departamento D.

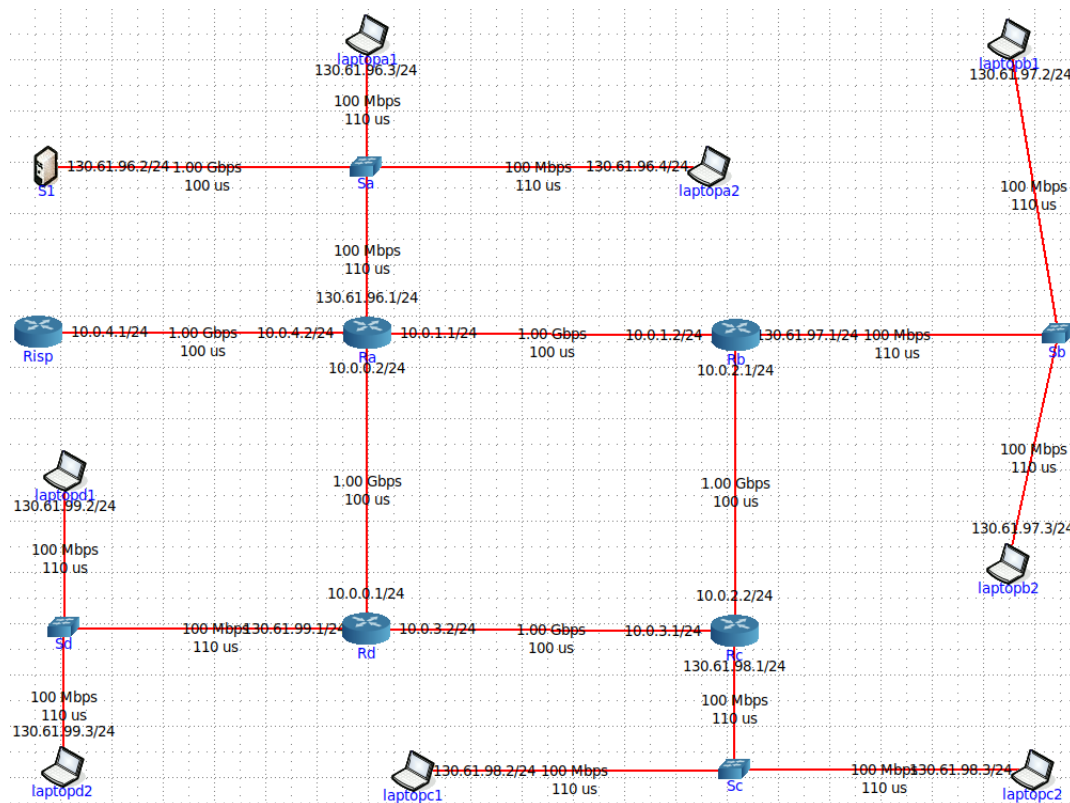


Figure 29: Topologia da rede

### 3.3.2 Alinea 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: 255.255.255.0.

Uma vez que os endereços com os bits todos a zero e a um não são usáveis, podemos ligar a  $2^8 - 2 = 254$  endereços.

### 3.3.3 Alinea 3

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

Para verificar a conectividade entre as várias redes locais, enviamos pings desde o servidor do departamento A até um laptop de cada departamento.

```
root@S1:/tmp/pycore.43407/S1.conf# ping 130.61.96.4
PING 130.61.96.4 (130.61.96.4) 56(84) bytes of data.
64 bytes from 130.61.96.4: icmp_seq=1 ttl=64 time=2.52 ms
64 bytes from 130.61.96.4: icmp_seq=2 ttl=64 time=0.964 ms
64 bytes from 130.61.96.4: icmp_seq=3 ttl=64 time=1.86 ms
^C
--- 130.61.96.4 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.964/1.785/2.524/0.640 ms
root@S1:/tmp/pycore.43407/S1.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=2 ttl=63 time=4.49 ms
64 bytes from 10.0.4.1: icmp_seq=3 ttl=63 time=1.50 ms
64 bytes from 10.0.4.1: icmp_seq=4 ttl=63 time=3.30 ms
64 bytes from 10.0.4.1: icmp_seq=5 ttl=63 time=1.68 ms
^C
--- 10.0.4.1 ping statistics ---
5 packets transmitted, 4 received, 20% packet loss, time 4016ms
rtt min/avg/max/mdev = 1.507/2.746/4.492/1.230 ms
root@S1:/tmp/pycore.43407/S1.conf# ping 130.61.97.2
PING 130.61.97.2 (130.61.97.2) 56(84) bytes of data.
64 bytes from 130.61.97.2: icmp_seq=1 ttl=62 time=4.57 ms
64 bytes from 130.61.97.2: icmp_seq=2 ttl=62 time=5.06 ms
64 bytes from 130.61.97.2: icmp_seq=3 ttl=62 time=1.58 ms
64 bytes from 130.61.97.2: icmp_seq=4 ttl=62 time=3.43 ms
^C
--- 130.61.97.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 1.580/3.665/5.066/1.341 ms
root@S1:/tmp/pycore.43407/S1.conf# ping 130.61.98.3
PING 130.61.98.3 (130.61.98.3) 56(84) bytes of data.
64 bytes from 130.61.98.3: icmp_seq=1 ttl=61 time=3.72 ms
64 bytes from 130.61.98.3: icmp_seq=2 ttl=61 time=2.58 ms
64 bytes from 130.61.98.3: icmp_seq=3 ttl=61 time=3.89 ms
64 bytes from 130.61.98.3: icmp_seq=4 ttl=61 time=2.19 ms
^C
--- 130.61.98.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3006ms
rtt min/avg/max/mdev = 2.131/3.089/3.898/0.718 ms
root@S1:/tmp/pycore.43407/S1.conf# ping 130.61.99.2
PING 130.61.99.2 (130.61.99.2) 56(84) bytes of data.
64 bytes from 130.61.99.2: icmp_seq=1 ttl=62 time=6.20 ms
64 bytes from 130.61.99.2: icmp_seq=2 ttl=62 time=9.28 ms
64 bytes from 130.61.99.2: icmp_seq=3 ttl=62 time=4.17 ms
64 bytes from 130.61.99.2: icmp_seq=4 ttl=62 time=3.04 ms
^C
--- 130.61.99.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 3.046/5.423/9.283/2.002 ms
root@S1:/tmp/pycore.43407/S1.conf# ping 10.0.2.2
PING 10.0.2.2 (10.0.2.2) 56(84) bytes of data.
64 bytes from 10.0.2.2: icmp_seq=1 ttl=62 time=2.73 ms
64 bytes from 10.0.2.2: icmp_seq=2 ttl=62 time=2.13 ms
64 bytes from 10.0.2.2: icmp_seq=3 ttl=62 time=7.87 ms
64 bytes from 10.0.2.2: icmp_seq=4 ttl=62 time=2.60 ms
^C
--- 10.0.2.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3007ms
rtt min/avg/max/mdev = 2.131/3.836/7.878/2.344 ms
```

Figure 30: Teste de conectividade

## 4 Conclusão

Este trabalho prático serviu de complemento às aulas teóricas e ajudou a consolidar a matéria lecionada nas mesmas.

Através da máquina virtual disponibilizada, tivemos a oportunidade de desenvolver as nossas capacidades de construção de Topologias CORE.

Relativamente ao capítulo de IP: Internet Protocol, ficamos a perceber melhor como é que o TTL funciona uma vez que tivemos a oportunidade de analisar vários exemplos, assim como a importância de tentar ter o menor TTL possível, mas que consiga chegar ao destino desejado.

Quanto ao formato de um datagrama IP, a par das aulas teóricas, identificamos os dois campos constituintes no datagrama: campo de dados (payload) e cabeçalho.

Averiguamos também a fragmentação de datagramas, e consequentemente, fomos confrontados com a importância do campo relativo à fragmentation, com respetivas flags (reserved bit, don't fragment, *more fragments* e fragment offset) (IPv4).

Relembramos ainda a definição de endereços públicos e privados e de switches.

Analisamos tabelas de encaminhamento e recordamos a definição de encaminhamento estático e dinâmico. Implicitamente, os conceitos de classfull e classless também foram utilizados.

Também trabalhamos com a definição de rotas estáticas e manipulação das mesmas.

O conceito de subnetting foi posto à prova com a necessidade da divisão de endereços e a importância das máscaras de rede foi também, mais uma vez, realçada.

Resumindo, basicamente todo o capítulo de Protocolo IP foi abrangido e lembrado, e os conceitos inerentes ao mesmo foram consolidados.