



UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

TP2: Protocolo IPv4
Datagramas IP e Fragmentação
PL12 – Grupo 120

Francisco Peixoto (a84668) José Fernandes (a93163)
Henrique Parola (a93325)

Março 2022

Conteúdo

1	Introdução	1
2	Parte I	2
2.1	Secção 1	2
2.2	Secção 2	5
2.3	Secção 3	8
3	Parte II	12
3.1	Secção 1	12
3.2	Secção 2	18
3.3	Secção 3	26
4	Conclusão	30

1 Introdução

Este trabalho foi realizado no âmbito da disciplina de Redes de Computadores da Licenciatura em Engenharia Informática da universidade do Minho.

O projeto foi dividido em duas partes. A primeira parte consiste em perguntas e respostas a respeito do protocolo IPv4, mais precisamente sobre Datagramas IP e Fragmentação. A segunda parte também se baseia em respostas a perguntas sobre o IPv4, mas sobre os temas de Endereçamento e Encaminhamento IP.

As ferramentas utilizadas para o desenvolvimento do trabalho foram nomeadamente: o **Core** (para a criação da topologia das redes) e o *Wireshark* (para a captura dos pacotes trocados entre os sistemas da topologia).

O presente relatório inicialmente apresenta as questões e respostas relativas à primeira parte do trabalho, sempre com imagens ilustrativas da rede desenvolvida e comprovativas de alguns resultados obtidos. A segunda parte reside sobre as questões e respostas sobre os temas de Endereçamento e Encaminhamento IP. Por fim será apresentada uma análise crítica do trabalho desenvolvido, relatando as principais dificuldades e detalhes importantes obtidos longo do trabalho.

2 Parte I

2.1 Secção 1

Topologia CORE para verificar o comportamento do traceroute:

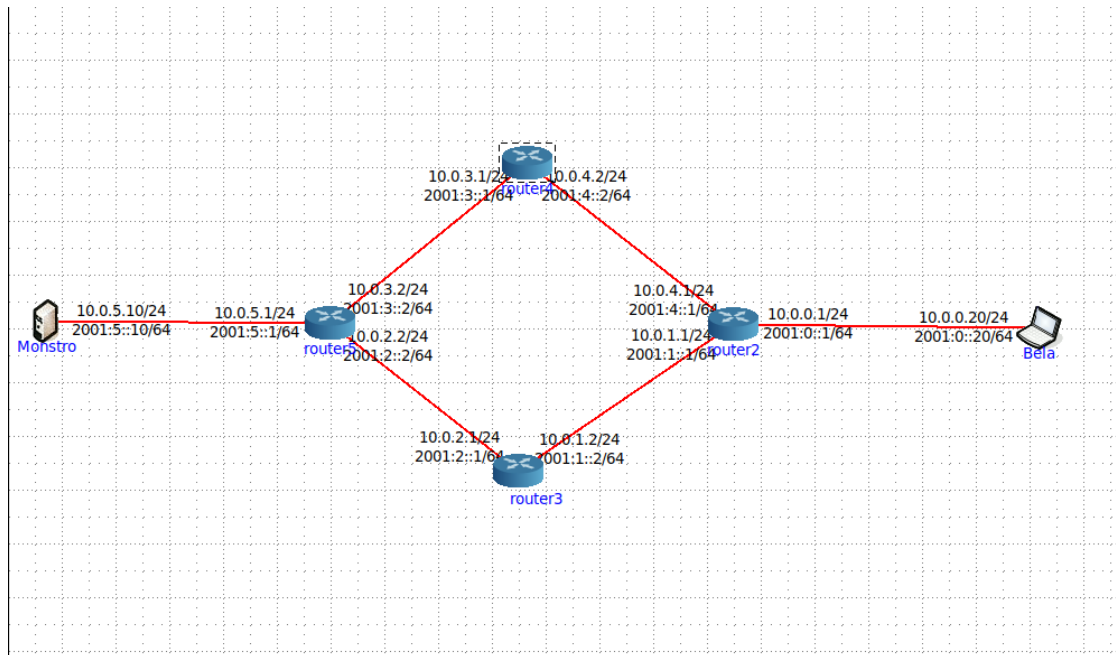


Figura 1: Topologia CORE, Parte I - Secção 1

A) Active o wireshark ou o tcpdump no host Bela. Numa shell de Bela execute o comando `traceroute -I` para o endereço IP do Monstro.

```
root@Bela:/tmp/pycore.37707/Bela.conf# traceroute -I 10.0.5.10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.124 ms  0.020 ms  0.014 ms
 2  10.0.1.2 (10.0.1.2)  0.042 ms  0.024 ms  0.021 ms
 3  10.0.2.2 (10.0.2.2)  0.049 ms  0.028 ms  0.027 ms
 4  10.0.5.10 (10.0.5.10)  0.090 ms  0.036 ms  0.033 ms
root@Bela:/tmp/pycore.37707/Bela.conf#
```

Figura 2: Resultado do Traceroute do portátil Bela ao Monstro

B) Registe e analise o tráfego ICMP enviado pelo sistema Bela e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=1/256, ttl=1 (no response found!)
1...	138.667...	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=2/512, ttl=1 (no response found!)
1...	138.667...	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=3/768, ttl=1 (no response found!)
1...	138.667...	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=4/1024, ttl=2 (no response found!)
1...	138.667...	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=5/1280, ttl=2 (no response found!)
1...	138.667...	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=6/1536, ttl=2 (no response found!)
1...	138.667...	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=7/1792, ttl=3 (no response found!)
1...	138.667...	10.0.2.2	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=8/2048, ttl=3 (no response found!)
1...	138.667...	10.0.2.2	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=9/2304, ttl=3 (no response found!)
1...	138.667...	10.0.2.2	10.0.0.20	ICMP	102 Time-to-live exceeded	(Time to live exceeded in transit)
1...	138.667...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=10/2560, ttl=4 (reply in 125)
1...	138.668...	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x001c, seq=10/2560, ttl=61 (request in 124)
1...	138.668...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=11/2816, ttl=4 (reply in 127)
1...	138.668...	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x001c, seq=11/2816, ttl=61 (request in 126)
1...	138.668...	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request	id=0x001c, seq=12/3072, ttl=4 (reply in 129)
1...	138.668...	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply	id=0x001c, seq=12/3072, ttl=61 (request in 128)

Figura 3: Análise do tráfego ICMP enviado pelo sistema Bela no Wireshark

Os resultados obtidos estão de acordo com o comportamento esperado do *traceroute*. Isto porque, como pode ser visto na figura 2, podemos encontrar os seguintes comportamentos:

- Pacotes com o mesmo **TTL** foram enviados três vezes pelo *host* Bela;
- Todos os pacotes com o valor do campo **TTL** inferior a quatro foram respondidos com o erro de *Time-to-live exceeded* pelo protocolo ICMP.
- Apenas os pacotes com **TTL** igual a quatro foram respondidos pelo servidor Monstro.

Os comportamentos encontrados estão de acordo com a utilização do *traceroute* no contexto de nossa topologia, dado ser necessário a realização de no mínimo quatro saltos para um pacote que seja enviado pelo *host* Bela alcançar o servidor Monstro.

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

O valor mínimo do TTL para alcançar o servidor monstro é quatro. Para verificar na prática este resultado, foi realizado um *ping* com controlo do campo TTL para o servidor Monstro. Nas imagens a seguir será mostrado que o *ping* foi realizado com sucesso quando o TTL era 4 e insucesso quando era 3.

```

root@Bela:/tmp/pycore,37707/Bela.conf# ping -t 4 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=0.115 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=0.122 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=0.124 ms

```

Figura 4: Pacotes enviados com TTL abaixo do valor mínimo

```

root@Bela:/tmp/pycore,37707/Bela.conf# ping -t 3 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
From 10.0.2.2 icmp_seq=1 Time to live exceeded
From 10.0.2.2 icmp_seq=2 Time to live exceeded
From 10.0.2.2 icmp_seq=3 Time to live exceeded

```

Figura 5: Pacotes enviados com o TTL mínimo

D) Calcule o valor médio do tempo de ida-e-volta (RTT - Round-Trip Time) obtido no acesso ao servidor. Para melhorar a média, poderá alterar o número pacotes de prova com a opção `-q`.

Utilizando uma amostra de 10 pacotes enviados para o servidor obteve-se uma média de 0.037ms de tempo de ida-e-volta no acesso ao servidor.

```

root@Bela:/tmp/pycore,37707/Bela.conf# traceroute -I 10.0.5.10 -q 10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.070 ms  0.019 ms  0.014 ms  0.013 ms  0.015 ms  0.014 ms  * * * *
 2  10.0.1.2 (10.0.1.2)  0.035 ms  0.023 ms  0.122 ms  0.057 ms  0.023 ms  0.022 ms  * * * *
 3  10.0.2.2 (10.0.2.2)  0.044 ms  0.030 ms  0.028 ms  0.028 ms  0.028 ms  0.027 ms  * * * *
 4  10.0.5.10 (10.0.5.10)  0.043 ms  0.034 ms  0.062 ms  0.038 ms  0.033 ms  0.032 ms  0.032 ms  0.032 ms  0.031 ms  0.033 ms

```

Figura 6: Traceroute para o servidor Monstro

E) O valor médio do atraso num sentido (One-Way Delay) poderia ser calculado com precisão dividindo o RTT por dois? O que torna difícil o cálculo desta métrica?

O valor médio do atraso num sentido é muitas vezes calculado como sendo o valor do RTT a dividir por dois. Porém há que realçar que isto seria apenas uma aproximação e, sem as garantias de que seja uma **boa** aproximação. Isto porque as redes de computadores são dinâmicas (rotas são alternadas constantemente) e assimétricas, principalmente pelo facto de haver ruído e imprevisibilidades na rede. Estes factores são oriundos das variadas condições de tráfego e características das diferentes redes envolvidas.

A razão pela qual é assumido a simples divisão do RTT por dois enunciada anteriormente, surge por conta do problema da sincronização de relógios (que seria a alternativa para encontrar um valor preciso do *One-Way Delay*. Caso, num cenário ideal, os relógios de diferente sistemas estivessem em perfeita sincronia, bastava o *receiver* responder um pacote enviado pelo *sender* com uma *timestamp* do momento de recebimento. Porém, como a sincronização dos relógios ainda é uma tarefa complexa de se garantir num ambiente computacional com enorme precisão, é preferível a alternativa simples de dividir o RTT por dois.

2.2 Secção 2

```
/home/jose/Universidade/3ano2sem/RC/PL/TP1/questao2_1.pcapng 97 total packets, 44 shown

No.      Time                Source            Destination      Protocol Length Info
 23 4.040734954      172.26.121.60      193.136.9.240    ICMP      74      Echo (ping) request id=0x04ee, seq=1/256, ttl=1 (no
response found!)
Frame 23: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_4c:df:ce (a0:51:0b:4c:df:ce), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.121.60, Dst: 193.136.9.240
Internet Control Message Protocol
  Type: 8 (Echo (ping) request)
  Code: 0
  Checksum: 0x7d8b [correct]
  [Checksum Status: Good]
  Identifier (BE): 1262 (0x04ee)
  Identifier (LE): 60932 (0xee04)
  Sequence Number (BE): 1 (0x0001)
  Sequence Number (LE): 256 (0x0100)
  [No response seen]
  Data (32 bytes)
0000 48 49 4a 4b 4c 4d 4e 50 51 52 53 54 55 56 57  HIJKLMNOPQRSTUVWXYZ
0010 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67  XYZ[\]^_`abcdefg
```

Figura 7: Primeira mensagem ICMP capturada

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

O endereço IP da interface ativa do computador é **172.26.121.60**. Isto pode ser verificado pelo campo *Source* na Figura 7.

B) Qual é o valor do campo protocolo? O que permite identificar?

O valor do campo protocolo é ICMP. Este campo permite identificar o protocolo que está sendo utilizado dentro do pacote IP. Neste caso o protocolo é o ICMP (cujo código protocolar é 0x01).

C) Quantos bytes tem o cabeçalho IPv4? Quantos bytes tem o campo de dados (payload) do datagrama? Como se calcula o tamanho do payload?

O cabeçalho IPv4 tem 20 bytes e o *payload* tem 40 bytes. O tamanho do *payload* pode ser obtido a partir do valor do *Total length* que é 60 (presente na Figura 8) subtraído pelo tamanho do cabeçalho.

```

Internet Protocol Version 4, Src: 172.26.121.60, Dst: 193.136.9.240
 0100 .... = Version: 4
 .... 0101 = Header Length: 20 bytes (5)
 Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   0000 00.. = Differentiated Services Codepoint: Default (0)
   .... ..00 = Explicit Congestion Notification: Not ECN-Capable Transport (0)
 Total Length: 60
 Identification: 0x695b (26971)
 Flags: 0x00
   0... .... = Reserved bit: Not set
   .0.. .... = Don't fragment: Not set
   ..0. .... = More fragments: Not set
   ...0 0000 0000 0000 = Fragment Offset: 0
 Time to Live: 1
   [Expert Info (Note/Sequence): "Time To Live" only 1]
     ["Time To Live" only 1]
     [Severity level: Note]
     [Group: Sequence]
 Protocol: ICMP (1)
 Header Checksum: 0x5f97 [validation disabled]
 [Header checksum status: Unverified]
 Source Address: 172.26.121.60
 Destination Address: 193.136.9.240

```

Figura 8: Primeira mensagem ICMP capturada - descrição do pacote IPV4

D) O datagrama IP foi fragmentado? Justifique.

O datagrama IP não foi fragmentado. Podemos concluir isto por conta da *flag More Fragments* estar a zero e o *offset* estar também a zero. Como a *flag More Fragments* a zero indica que o pacote é o último "fragmento" e *offset* a zero indica que o pacote é o primeiro "fragmento", então podemos concluir que ele é único e por tanto não fragmentado.

```

Flags: 0x00
 0... .... = Reserved bit: Not set
 .0.. .... = Don't fragment: Not set
 ..0. .... = More fragments: Not set
 ...0 0000 0000 0000 = Fragment Offset: 0

```

Figura 9: Primeira mensagem ICMP capturada - descrição da fragmentação

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 são o TTL, o identificador e o *checksum*. O TTL varia justamente por se tratar do mecanismo do *traceroute* de a cada três pacotes incrementar o TTL. Já o identificador varia por não haver fragmentação. Como o propósito deste identificador é auxiliar na remontagem de pacotes fragmentados que acabam por receber o mesmo identificador, era de se esperar a sua variação por estarmos a lidar com pacotes independentes um dos outros. O *checksum* variar é uma consequência da variação do identificador e do TTL.

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

Os valores do campo Identificação são todos diferentes. Os valores do campo TTL são incrementados a cada três pacotes.

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 não permanece constante para todas as mensagens. À semelhança do campo TTL dos pacotes enviados pelo nosso *host*, o campo TTL nas mensagens ICMP TTL exceeded recebidas são constantes por cada três pacotes. Por outras palavras, como o *traceroute* envia três pacotes com TTL iguais para cada *destination*, os pacotes de resposta (nessas situações de ter excedido o TTL) de cada *destination* também possuem mesmo TTL.

A variação dá-se por conta do decremento que cada *router* faz ao TTL até chegar ao *host*. E para além disto, como o TTL *default* de uma mensagem ICMP é 255, podemos concluir que os resultados estão coesos, dado três pacotes terem sido capturados com TTL 255, outros três com 254 e finalmente outros três com 253.

Por outro lado, num situação real este resultado poderia não ter sido observado. O padrão de valores de TTL observado poderia não se verificar uma vez que o caminho até ao *router*(caminho de *echo*) poderá ser diferente do caminho do *router* até ao *host*(caminho de TTL exceeded). Desta forma os saltos do caminho de *echo* podem ser diferentes ao do caminho de TTL exceeded e portanto o resultado que observássemos seria diferente e mais irregular.

2.3 Secção 3

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

Uma vez que o **MTU** da interface a partir do qual o tráfego foi capturado é de 1500 *bytes*, isto significa a interface apenas consegue enviar 1500 *bytes* de *payload* por trama. Ou seja, para que os 4120 *bytes* sejam transmitidos eles vão ter que ser divididos por vários pacotes com tamanho de *payload* menor ou igual ao **MTU**, ou seja, houve a necessidade de fragmentar tal pacote.

```
wlp0s20f3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
```

Figura 10: Informações interface de captura

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?

```
Frame 9: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_4c:df:ce (a0:51:0b:4c:df:ce), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.121.60, 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: 0x948b (38027)
  Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0x0ec7 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.121.60
  Destination Address: 193.136.9.240
  [Reassembled IPv4 in frame: 11]
```

Figura 11: Primeiro fragmento segmentado

A informação no cabeçalho que indica que o datagrama foi segmentado é a *flag* **More segments** estar com o valor 1. A informação que nos leva a concluir de que se trata do primeiro fragmento é o campo do *offset* estar à zero. O tamanho deste datagrama IP é de 1500 *bytes*. Calculamos este valor a partir da subtração do tamanho total Ethernet *frame* pelo seu cabeçalho (14 *bytes*). Este valor também está indicado diretamente no campo *Total length* indicado na figura 11.

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?

```
Frame 10: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_4c:df:ce (a0:51:0b:4c:df:ce), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.121.60, 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: 0x948b (38027)
    Flags: 0x20, More fragments
      0... .... = Reserved bit: Not set
      .0... .... = Don't fragment: Not set
      ..1. .... = More fragments: Set
    ...0 0101 1100 1000 = Fragment Offset: 1480
    Time to Live: 1
    Protocol: ICMP (1)
    Header Checksum: 0x0e0e [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.121.60
    Destination Address: 193.136.9.240
    [Reassembled IPv4 in frame: 11]
```

Figura 12: Primeiro fragmento segmentado

A informação do cabeçalho IP que indica que não se trata do 1º fragmento é justamente o *offset* estar diferente de zero. Podemos afirmar que há mais fragmentos justamente pela *flag More fragments* estar com o valor 1.

D) Quantos fragmentos foram criados a partir do datagrama original?

Foram criados três fragmentos a partir do datagrama original. Como pode ser visto na próxima página, são mostradas três figuras, cada uma com a captura de cada fragmento. Nas primeiras duas capturas a *flag More fragments* está com o valor 1 enquanto na terceira captura está com o valor zero (indicando que é o último fragmento).

```

Frame 9: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_4c:df:ce (a0:51:0b:4c:df:ce), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.121.60, 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: 0x948b (38027)
  Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
  ...0 0000 0000 0000 = Fragment Offset: 0
  Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0x0ec7 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.121.60
  Destination Address: 193.136.9.240
  [Reassembled IPv4 in frame: 11]

```

Figura 13: Primeiro fragmento segmentado

```

Frame 10: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_4c:df:ce (a0:51:0b:4c:df:ce), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.121.60, 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: 0x948b (38027)
  Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
  ...0 0101 1100 1000 = Fragment Offset: 1400
  Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0x0e0e [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.121.60
  Destination Address: 193.136.9.240
  [Reassembled IPv4 in frame: 11]

```

Figura 14: Segundo fragmento segmentado

```

No.    Time          Source            Destination       Protocol Length Info
 11 0.841994040 172.26.121.60    193.136.9.240    ICMP      1174  Echo (ping) request id=0x0529, seq=1/256, ttl=1 (no
response found!)
Frame 11: 1174 bytes on wire (9392 bits), 1174 bytes captured (9392 bits) on interface wlp0s20f3, id 0
Ethernet II, Src: IntelCor_4c:df:ce (a0:51:0b:4c:df:ce), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.121.60, 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: 1160
  Identification: 0x948b (38027)
  Flags: 0x01
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
  ...0 1011 1001 0000 = Fragment Offset: 2960
  Time to Live: 1
  Protocol: ICMP (1)
  Header Checksum: 0x2ea9 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 172.26.121.60
  Destination Address: 193.136.9.240
  [3 IPv4 Fragments (4100 bytes): #9(1480), #10(1480), #11(1140)]
Internet Control Message Protocol

```

Figura 15: Terceiro fragmento segmentado

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 mudam no cabeçalho IP entre os diferentes fragmentos são os *offsets*. A partir deles podemos ter controlo da posição na qual o *payload* do pacote se encontrará na reconstrução do pacote que foi segmentado. Para além disto, outro factor relevante para a reconstrução do pacote é o campo de **identificação** presente no cabeçalho IP. Fragmentos com o mesmo valor neste campo são constituintes de um mesmo pacote.

F) Verifique o processo de fragmentação através de um processo de cálculo.

Inicialmente o *traceroute* gera um datagrama IP com 4120 bytes de tamanho total (*payload* + *header*). Porém, como o **MTU** da interface de captura é 1500 *bytes*, significa que os pacotes IP a serem enviados terão de ter 1500 *bytes* de tamanho total. Ou seja, como o tamanho do *header* IP é 20 *bytes*, então o *payload* de 4100 *bytes* a ser enviado deverá ser dividido em três segmentos. Os dois primeiros segmentos terão de ter 1480 *bytes* de *payload* e 20 *bytes* de *header* (para cada um dos pacotes). Resta por tanto um terceiro segmento, mas dessa vez sem ter 1500 *bytes* e sim 1160 *bytes* (20 *bytes* para o *header* e 1140 *bytes* para o restante do *payload*, obtido através da conta: **4100 - 2960**). Na conta anterior, **2960** corresponde ao *payload* já enviado e **4100** o *payload* total inicial. Deste modo a diferença entre estes valores corresponde ao resto do *payload* total a enviar no terceiro fragmento. Estes valores podem ser obtido através da expressão:

TP = Tamanho total do *payload* a enviar; **TPM** = Tamanho do *payload* do MTU.

$$\text{Quantidade de fragmentos} = \frac{TP}{TPM} \text{ (arredondado para cima)}$$

$$\text{payload do último fragmento} = TP \% TPM$$

G) Escreva uma expressão lógica que permita detetar o último fragmento correspondente ao datagrama original.

Supondo inicialmente que o primeiro fragmento correspondente ao datagrama original tem no campo de **Identificação** o valor *X*, então para detetar o último fragmento basta procurar por um pacote cujo cabeçalho do IP tenha o tal campo de identificação *X* e para além disto a *flag* **More fragments** com o valor 0.

3 Parte II

3.1 Secção 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.

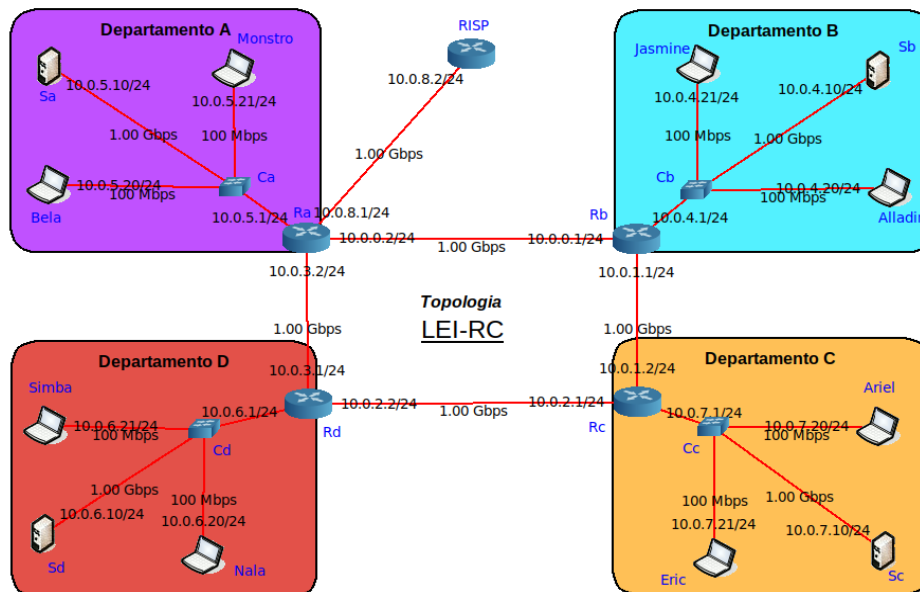


Figura 16: Topologia LEI-RC

Nesta topologia os *laptops* e servidores apresentam uma única interface Ethernet. Segue uma tabela com os endereços IP's das interfaces de cada *laptop* e servidor:

Descrição dos endereços da topologia			
Equipamento	Endereço IP	Rede	Máscara (notação decimal/ CIDR)
Monstro	10.0.5.21	10.0.5.0	255.255.255.0(/24)
Bela	10.0.5.20	10.0.5.0	
Servidor A	10.0.5.10	10.0.5.0	
Jasmine	10.0.4.21	10.0.4.0	
Alladin	10.0.4.20	10.0.4.0	255.255.255.0(/24)
Servidor B	10.0.4.10	10.0.4.0	
Ariel	10.0.7.20	10.0.7.0	
Eric	10.0.7.21	10.0.7.0	
Servidor C	10.0.7.10	10.0.7.0	255.255.255.0(/24)
Simba	10.0.6.21	10.0.6.0	
Nala	10.0.6.20	10.0.6.0	
Servidor D	10.0.6.10	10.0.6.0	

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

Todos os endereços atribuídos pelo CORE são endereços privados visto que pertencem ao bloco de endereços IP da Classe A, reservados pela IANA (Internet Assigned Numbers Authority) para internets privadas e que se situam entre 10.0.0.0 - 10.255.255.255/8, conforme pode ser consultado no RFC 1918.

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

Não é atribuído um endereço aos switches porque estes são equipamentos da segunda camada do modelo OSI (Data Link), cuja funcionalidade é interligar equipamentos de uma rede, não sendo por isso considerados como *hosts* na rede.

D) Usando o comando *ping* certifique-se que existe conectividade IP interna a cada departamento (e.g. entre um laptop e o servidor respetivo).

```
root@Monstro:/tmp/pycore.36003/Monstro.conf# ping -c 5 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_seq=1 ttl=64 time=0.414 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=0.367 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=64 time=0.127 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=64 time=0.167 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=64 time=0.083 ms

--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4101ms
rtt min/avg/max/mdev = 0.083/0.231/0.414/0.133 ms
```

Figura 17: Conexão interna departamento A

```

root@Alladin:/tmp/pycore.36003/Alladin.conf# ping -c 5 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=64 time=0.417 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=64 time=0.071 ms
64 bytes from 10.0.4.10: icmp_seq=5 ttl=64 time=0.078 ms

--- 10.0.4.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4080ms
rtt min/avg/max/mdev = 0.065/0.140/0.417/0.138 ms

```

Figura 18: Conexão interna departamento B

```

root@Eric:/tmp/pycore.36003/Eric.conf# ping -c 5 10.0.7.10
PING 10.0.7.10 (10.0.7.10) 56(84) bytes of data.
64 bytes from 10.0.7.10: icmp_seq=1 ttl=64 time=0.402 ms
64 bytes from 10.0.7.10: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.0.7.10: icmp_seq=3 ttl=64 time=0.071 ms
64 bytes from 10.0.7.10: icmp_seq=4 ttl=64 time=0.123 ms
64 bytes from 10.0.7.10: icmp_seq=5 ttl=64 time=0.076 ms

--- 10.0.7.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4101ms
rtt min/avg/max/mdev = 0.071/0.149/0.402/0.127 ms

```

Figura 19: Conexão interna departamento C

```

root@Simba:/tmp/pycore.36003/Simba.conf# ping -c 5 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=64 time=0.549 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=64 time=0.094 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=64 time=0.074 ms
64 bytes from 10.0.6.10: icmp_seq=5 ttl=64 time=0.075 ms

--- 10.0.6.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4089ms
rtt min/avg/max/mdev = 0.067/0.171/0.549/0.188 ms

```

Figura 20: Conexão interna departamento D

A partir das figuras apresentadas demonstra-se que foi possível obter um pedido e uma resposta entre um portátil e um servidor de cada departamento e portanto que existe conectividade IP interna em cada departamento.

E) Execute o número mínimo de comandos *ping* que lhe permite verificar a existência de conectividade IP entre departamentos.

Para testar a conectividade entre departamentos era necessário verificar se os *routers* tinham a informação necessária para encaminhar os pacotes para todos os departamentos. Para testar foi feito o comando *ping*. Este teste testa se os *routers* de entrada dos departamentos em questão conseguem encaminhar pacotes entre estes dois departamentos. Isto porque o comando que é feito para o Departamento X testa a capacidade dos routers de conseguirem encaminhar para o Destino na rede X e a resposta *echo* vai ter destino o Departamento Y, que fez o comando, que testa a capacidade dos routers conseguirem encaminhar para departamento Y. Desta forma foram feitos os seguintes testes:

1. *Ping* do departamento A para os restantes.

Conhecimento confirmado após 1 Ping	
<i>Router</i>	Conhecimento confirmado
A	A,B,C,D
B	A,B
C	A,C
D	A,D

2. *Ping* do departamento B para o departamento C e D

Conhecimento confirmado após 2 Ping's	
<i>Router</i>	Conhecimento confirmado
A	A,B,C,D
B	A,B,C,D
C	A,C,B
D	A,D,B

3. *Ping* do departamento C para o departamento D

Conhecimento confirmado após 3 Ping's	
<i>Router</i>	Conhecimento confirmado
A	A,B,C,D
B	A,B,C,D
C	A,C,B,D
D	A,D,B,C

De seguida serão mostrados os *prints* dos comandos Pings's realizados para a construção das tabelas anteriores:

```
root@Bela:/tmp/pycore.36003/Bela.conf# ping -c 5 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=62 time=0.517 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=62 time=0.101 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=62 time=0.112 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=62 time=0.095 ms
64 bytes from 10.0.4.10: icmp_seq=5 ttl=62 time=0.189 ms

--- 10.0.4.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4086ms
rtt min/avg/max/mdev = 0.095/0.202/0.517/0.160 ms
```

Figura 21: Teste *ping* entre departamento A e B

```
root@Bela:/tmp/pycore.36003/Bela.conf# ping -c 5 10.0.7.10
PING 10.0.7.10 (10.0.7.10) 56(84) bytes of data.
64 bytes from 10.0.7.10: icmp_seq=1 ttl=61 time=0.490 ms
64 bytes from 10.0.7.10: icmp_seq=2 ttl=61 time=0.135 ms
64 bytes from 10.0.7.10: icmp_seq=3 ttl=61 time=0.136 ms
64 bytes from 10.0.7.10: icmp_seq=4 ttl=61 time=0.134 ms
64 bytes from 10.0.7.10: icmp_seq=5 ttl=61 time=0.221 ms

--- 10.0.7.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4081ms
rtt min/avg/max/mdev = 0.134/0.223/0.490/0.137 ms
```

Figura 22: Teste *ping* entre departamento A e C

```
root@Bela:/tmp/pycore.36003/Bela.conf# ping -c 5 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=62 time=0.481 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=62 time=0.178 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=62 time=0.120 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=62 time=0.102 ms
64 bytes from 10.0.6.10: icmp_seq=5 ttl=62 time=0.114 ms

--- 10.0.6.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4103ms
rtt min/avg/max/mdev = 0.102/0.199/0.481/0.143 ms
```

Figura 23: Teste *ping* entre departamento A e D

```

root@Alladin:/tmp/pycore.36003/Alladin.conf# ping -c 5 10.0.7.10
PING 10.0.7.10 (10.0.7.10) 56(84) bytes of data.
64 bytes from 10.0.7.10: icmp_seq=1 ttl=62 time=0.456 ms
64 bytes from 10.0.7.10: icmp_seq=2 ttl=62 time=0.114 ms
64 bytes from 10.0.7.10: icmp_seq=3 ttl=62 time=0.111 ms
64 bytes from 10.0.7.10: icmp_seq=4 ttl=62 time=0.111 ms
64 bytes from 10.0.7.10: icmp_seq=5 ttl=62 time=0.115 ms

--- 10.0.7.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4075ms
rtt min/avg/max/mdev = 0.111/0.181/0.456/0.137 ms

```

Figura 24: Teste *ping* entre departamento B e C

```

root@Alladin:/tmp/pycore.36003/Alladin.conf# ping -c 5 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=61 time=0.496 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=61 time=0.138 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=61 time=0.133 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=61 time=0.140 ms
64 bytes from 10.0.6.10: icmp_seq=5 ttl=61 time=0.144 ms

--- 10.0.6.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4076ms

```

Figura 25: Teste *ping* entre departamento B e D

```

root@Ariel:/tmp/pycore.36003/Ariel.conf# ping -c 5 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=62 time=0.458 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=62 time=0.108 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=62 time=0.110 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=62 time=0.110 ms
64 bytes from 10.0.6.10: icmp_seq=5 ttl=62 time=0.155 ms

--- 10.0.6.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4101ms
rtt min/avg/max/mdev = 0.108/0.188/0.458/0.136 ms

```

Figura 26: Teste *ping* entre departamento C e D

F) Verifique se existe conectividade IP do portátil Bela para o router de acesso RISP.

Pode-se testar esta conectividade através de um comando Ping entre o portátil Bela e o *router* de acesso RISP.

```
root@Bela:/tmp/pycore.36003/Bela.conf# ping -c 5 10.0.8.2
PING 10.0.8.2 (10.0.8.2) 56(84) bytes of data.
64 bytes from 10.0.8.2: icmp_seq=1 ttl=63 time=0.457 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=63 time=0.094 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=63 time=0.097 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=63 time=0.108 ms
64 bytes from 10.0.8.2: icmp_seq=5 ttl=63 time=0.095 ms

--- 10.0.8.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4079ms
rtt min/avg/max/mdev = 0.094/0.170/0.457/0.143 ms
```

Figura 27: Teste ping entre portátil Bela e router de acesso RISP

3.2 Secção 2

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@Bela:/tmp/pycore.41423/Bela.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.5.1 0.0.0.0 UG 0 0 0 eth0
10.0.5.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@Bela:/tmp/pycore.41423/Bela.conf#
```

Figura 28: Tabela de encaminhamento portátil Bela

```
root@Ra:/tmp/pycore.41423/Ra.conf# netstat -nr
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 eth1
10.0.1.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth1
10.0.2.0 10.0.3.1 255.255.255.0 UG 0 0 0 eth0
10.0.3.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.4.0 10.0.0.1 255.255.255.0 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 10.0.3.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 10.0.3.1 255.255.255.0 UG 0 0 0 eth0
10.0.8.0 0.0.0.0 255.255.255.0 U 0 0 0 eth3
root@Ra:/tmp/pycore.41423/Ra.conf#
```

Figura 29: Tabela de encaminhamento *router* Ra

A primeira coluna do *output* do comando *netstat -rn* corresponde a rede destino. A segunda coluna (*Gateway*) corresponde ao "próximo salto" (endereço da interface) para o qual os pacotes devem ser encaminhados (tendo em conta a rede destino). A coluna *Genmask* corresponde a máscara da rede destino.

As *Flags* encontradas são de dois tipos:

- **U** - A rota é utilizável. Caso contrário ela seria inacessível e não seria usada para roteamento;
- **G** - De acordo com o manual do *netstat*: "destination requires forwarding by intermediary". Caso essa *flag* não estivesse presente, poder-se-ia dizer que a rota está diretamente ligada ao destino.

A coluna *MSS* é relativa as conexões TCP feitas sobre a rota. Indica o "Maximum Segment Size" do datagrama a ser construído pelo *kernel* para ser enviado na respetiva rota (o valor do *MSS* não conta o cabeçalho IP nem TCP, conta apenas o *payload* do TCP). Este tamanho corresponde ao *MTU* subtraído os *headers* do protocolo IP e TCP.

A coluna *Window* representa o "Default Window Size" para conexões TCP na rota indicada, que representa a quantidade máxima de dados que pode ser recebido por um segmento TCP. A coluna *IRTT* indica o "Initial Round Trip Time", este valor auxilia o *kernel* a fazer ajustes dinâmicos aos parâmetros TCP em conexões remotas que demonstram ser lentas a retornar respostas. Por fim a coluna *Iface* é o identificador da interface de saída da máquina local. Para as três colunas referidas anteriormente o valor estar à zero em todas as entradas significa que o valor utilizado é o *default*.

Tendo sido explicado os valores das colunas das tabelas, procederá então uma breve interpretação para vermos se os valores encontrados fazem sentido. Inicialmente vemos que para o portátil *Bela*, todas as entradas da tabela possuem o mesmo valor da coluna *Iface*. Isto está de acordo com o esperado, dado tal portátil possuir apenas uma interface Ethernet. Em contrapartida, para o *router Ra*, esta mesma coluna varia em 4 valores distintos. O que novamente está de acordo com a topologia, dado o *router Ra* possuir quatro interfaces Ethernet. Em relação a *flag G*, podemos confirmar seu significado teórico ao analisar a topologia. Tome-se como exemplo a tabela de encaminhamento do *router Ra*. Para as redes destino **10.0.0.0**, **10.0.3.0**, **10.0.5.0** e **10.0.8.0** tal *flag G* não está presente. E isto faz sentido, dado que essas são as quatro redes destino as quais não será preciso intermediários para os pacotes enviados alcançarem tais redes. Por outras palavras, estas são as quatro redes nas quais o *router Ra* possui rotas diretamente ligadas. O valor da máscara a **255.255.255.0** também vai de encontro a análise realizada no começo desta secção.

Em suma, a leitura da penúltima linha (por exemplo) da tabela de encaminhamento da figura 29 pode ser feita da seguinte maneira: "Um datagrama destinado à rede 10.0.7.0 será entregue na interface de endereço 10.0.3.1 saindo pela interface local eht0".

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 ou equivalente).

```
root@Bela:/tmp/pycore.43093/Bela.conf# ps -ax
  PID TTY          STAT TIME   COMMAND
    1 ?            S      0:00 vncnode -v -c /tmp/pycore.43093/Bela -l /tmp/pycore.
   71 pts/3      Ss      0:00 /bin/bash
   81 pts/3      R+      0:00 ps -ax
root@Bela:/tmp/pycore.43093/Bela.conf#
```

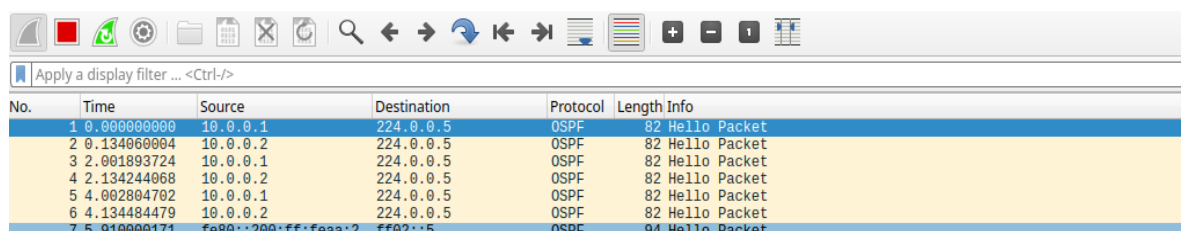
Figura 30: ps -ax output - portátil Bela

```
root@Ra:/tmp/pycore.43093/Ra.conf# ps -ax
  PID TTY          STAT TIME   COMMAND
    1 ?            S      0:00 vncnode -v -c /tmp/pycore.43093/Ra -l /tmp/pycore.43093/Ra.log -p /t
   75 ?            Ss      0:00 /usr/local/sbin/zebra -d
   81 ?            Ss      0:00 /usr/local/sbin/ospf6d -d
   85 ?            Ss      0:01 /usr/local/sbin/ospfd -d
  100 pts/1      Ss      0:00 /bin/bash
  111 pts/1      R+      0:00 ps -ax
root@Ra:/tmp/pycore.43093/Ra.conf#
```

Figura 31: ps -ax output - router Ra

Os processos com PID 75, 81 e 85 do *router* Ra indicam-nos que nesse sistema está a ser usado encaminhamento dinâmico. Isto porque estes processos mencionados correspondem a programas relativos a *routing manager*. Nas palavras da **gnu.org**, "Zebra é um *software* de roteamento multi-servidor que fornece protocolos de roteamento TCP/IP". Dentre estes protocolos temos: RIPv1, RIPv2, RIPv6, OSPF, OSPF6, dentre outros. Estes protocolos de encaminhamento são **protocolos dinâmicos**.

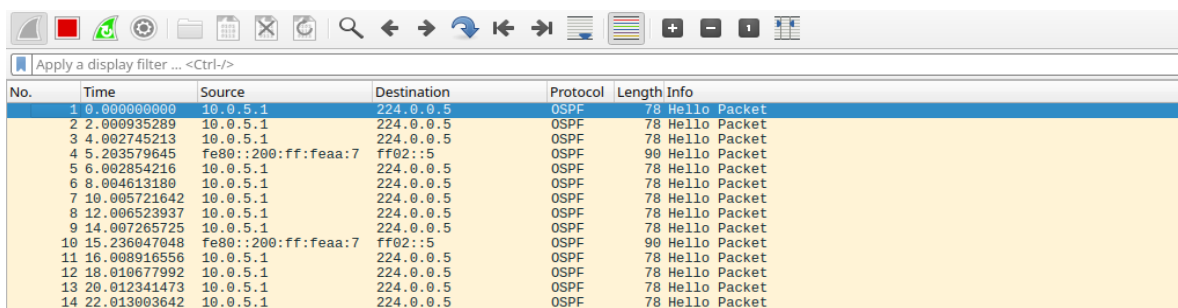
Para além disto, há ainda os *daemons* ospf6d e ospfd que gerenciam tabelas de roteamento. Este gerenciamento ocorre de acordo com o princípio de **estado de ligações**, onde novamente, estamos perante protocolos dinâmicos. Mas, adicionalmente, de acordo com a *man page* do ospf6d, os roteadores OSPF descobrem uns aos outros automaticamente por meio de *hello packets OSPF*. Desta maneira, abrimos o *Wireshark* para capturar o tráfego de uma das interfaces do Ethernet do *router* Ra e presenciamos o resultado esperado: foram capturados os tais pacotes mencionados, apoiando ainda mais que tal *router* utiliza encaminhamento dinâmico.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
2	0.134060004	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
3	2.001893724	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
4	2.134244068	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
5	4.002804702	10.0.0.1	224.0.0.5	OSPF	82	Hello Packet
6	4.134484479	10.0.0.2	224.0.0.5	OSPF	82	Hello Packet
7	5.910000171	fe80::200:ff:feaa:2	ff02::5	OSPF	94	Hello Packet

Figura 32: Captura de pacotes OSPF - router Ra

Porém há um ponto interessante a observar. Mostrar pacotes OSPF capturados, mostrados no *print* anterior, não é o suficiente para a conclusão que foi tirada sobre o *router* utilizar protocolo de encaminhamento dinâmico. O que leva-nos objetivamente a concluir que tal roteador utiliza o protocolo de estados de ligações OSPF é o facto dele ter sido responsável por ter enviado alguns desses pacotes OSPF (basta ver os pacotes capturados com *source* 10.0.0.2 que é a interface Ethernet selecionada para captura). Isto porque os *routers* OSPF comunicam via *multicast*, mais precisamente através do endereço *multicast* 224.0.0.5 (como pode ser visto no campo *destination* do *print* mostrado). Como um *multicast* funciona como um *broadcast* que é a transmissão de informação para múltiplos destinatários simultaneamente, significa que se for aberta uma captura no portátil Bela, é capaz de ser encontrado também estes *Hello packets*:



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
2	2.0009935289	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
3	4.002745213	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
4	5.203579645	fe80::200:ff:feaa:7	ff02::5	OSPF	90	Hello Packet
5	6.002854216	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
6	8.004613180	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
7	10.005721642	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
8	12.006523937	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
9	14.007265725	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
10	15.236047048	fe80::200:ff:feaa:7	ff02::5	OSPF	90	Hello Packet
11	16.008916556	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
12	18.010677992	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
13	20.012341473	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet
14	22.013903642	10.0.5.1	224.0.0.5	OSPF	78	Hello Packet

Figura 33: Captura de pacotes OSPF - portátil Bela

Porém tal portátil não está a utilizar encaminhamento dinâmico e consequentemente não está a fazer *multicast* dos tais pacotes mencionados como o *router* Ra. A captura mostra que tal portátil apenas foi alvo (*destination*) destes pacotes vindos da *source* 10.0.5.1 que, não por acaso, é uma das interfaces do *router* Ra.

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 SA. Use o comando `route delete` para o efeito. Que implicações tem esta medida para os utilizadores da LEI-RC que acedem ao servidor. Justifique.

```
root@Sa:/tmp/pycore.43093/Sa.conf# netstat -nr
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0        10.0.5.1        0.0.0.0         UG      0 0        0 eth0
10.0.5.0       0.0.0.0         255.255.255.0   U       0 0        0 eth0
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route del -net 0.0.0.0 netmask 0.0.0.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# netstat -nr
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.5.0       0.0.0.0         255.255.255.0   U       0 0        0 eth0
root@Sa:/tmp/pycore.43093/Sa.conf#
```

Figura 34: Remoção da *default route* do servidor SA

Como pode ser visto nas imagens a seguir, apenas máquinas do departamento A (departamento do servidor SA) obtém conectividade IP com o servidor SA. Os demais departamentos perderam sua conectividade:

```
root@Bela:/tmp/pycore.43093/Bela.conf# ping -c 5 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.
64 bytes from 10.0.5.10: icmp_seq=1 ttl=64 time=0.160 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=64 time=0.328 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=64 time=0.192 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=64 time=0.921 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=64 time=0.354 ms

--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4053ms
rtt min/avg/max/mdev = 0.160/0.391/0.921/0.275 ms
root@Bela:/tmp/pycore.43093/Bela.conf#
```

Figura 35: Teste de conectividade IP - Bela & Servidor SA

```
root@Jasmine:/tmp/pycore.43093/Jasmine.conf# ping -c 5 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.

--- 10.0.5.10 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4102ms

root@Jasmine:/tmp/pycore.43093/Jasmine.conf#
```

Figura 36: Teste de conectividade IP - Jasmine & Servidor SA

```
root@Ariel:/tmp/pycore.43093/Ariel.conf# ping -c 5 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.

--- 10.0.5.10 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4082ms

root@Ariel:/tmp/pycore.43093/Ariel.conf#
```

Figura 37: Teste de conectividade IP - Ariel & Servidor SA

```
root@Simba:/tmp/pycore.43093/Simba.conf# ping -c 5 10.0.5-10
ping: 10.0.5-10: Temporary failure in name resolution
root@Simba:/tmp/pycore.43093/Simba.conf# ping -c 5 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data.

--- 10.0.5.10 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4075ms

root@Simba:/tmp/pycore.43093/Simba.conf#
```

Figura 38: Teste de conectividade IP - Simba & Servidor SA

Mas para além disto, o próprio servidor também apenas consegue se conectar com máquinas que estão em seu departamento:

```

root@Sa:/tmp/pycore.43093/Sa.conf# ping -c 5 10.0.5.20
PING 10.0.5.20 (10.0.5.20) 56(84) bytes of data.
64 bytes from 10.0.5.20: icmp_seq=1 ttl=64 time=0.862 ms
64 bytes from 10.0.5.20: icmp_seq=2 ttl=64 time=0.308 ms
^C
--- 10.0.5.20 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.308/0.585/0.862/0.277 ms
root@Sa:/tmp/pycore.43093/Sa.conf# ping -c 5 10.0.7.10
ping: connect: Network is unreachable
root@Sa:/tmp/pycore.43093/Sa.conf# ping -c 5 10.0.6.20
ping: connect: Network is unreachable
root@Sa:/tmp/pycore.43093/Sa.conf# ping -c 5 10.0.4.21
ping: connect: Network is unreachable
root@Sa:/tmp/pycore.43093/Sa.conf# █

```

Figura 39: Teste de conectividade IP - Simba & Servidor SA

Este comportamento era o esperado. A rota por defeito é aquela que lida com qualquer tráfego destinado fora de sua rede. Por outras palavras, quando não é possível saber com exatidão (através da tabela de encaminhamento) para onde a informação deve ser enviada, é então utilizada a rota por defeito. Pode-se dizer que normalmente a rota por defeito está ligada a um *router* que conecta sua máquina a redes remotas.

Feita esta explicação, faz sentido que os *ping's* realizados por máquinas de fora do departamento A não tiveram sucesso. Isto porque, por mais que os pacotes tenham chegado ao servidor, este não foi capaz de enviar a resposta de volta, justamente por não haver entradas na tabela de encaminhamento que lidassem com algo de fora de sua rede. Isto pode ser comprovado através da seguinte captura de um *ping* entre o portátil Ariel e o servidor A (onde não foi detetada mensagens de resposta):

No.	Time	Source	Destination	Protocol	Length	Info
23	34.494621857	10.0.7.20	10.0.5.10	ICMP	98	Echo (ping) request id=0x0030, seq=1/256, ttl=61 (no respons...
24	35.516578191	10.0.7.20	10.0.5.10	ICMP	98	Echo (ping) request id=0x0030, seq=2/512, ttl=61 (no respons...
26	36.541093528	10.0.7.20	10.0.5.10	ICMP	98	Echo (ping) request id=0x0030, seq=3/768, ttl=61 (no respons...
27	37.565391648	10.0.7.20	10.0.5.10	ICMP	98	Echo (ping) request id=0x0030, seq=4/1024, ttl=61 (no respons...
29	38.589775580	10.0.7.20	10.0.5.10	ICMP	98	Echo (ping) request id=0x0030, seq=5/1280, ttl=61 (no respon...

Figura 40: Captura do tráfego do

Em contrapartida, *ping's* enviados pelo servidor a outra máquina do departamento A (ou vice-versa) tiveram sucesso. Isto sucedeu-se porque ambas as máquinas estão na **mesma rede** e por tanto haverá uma entrada na tabela de encaminhamento que vai dar *match* ao aplicar a máscara da rede. Nesses casos, o próximo salto a ser dado é para a interface **0.0.0.0**, o que significa que o pacote não precisa ser roteado e pode, por tanto, ser enviado diretamente ao destino na rede local. O processo que acontece nessa situação é um *ARP request*, onde será feito um *broadcast* a todos os elementos da rede com o intuito de obter o *MAC address* associado ao IP da máquina que se queira enviar o pacote (a máquina destino será a única a responder o *broadcast* e por isso será possível enviar o pacote a ela depois).

Para comprovar o que foi dito, segue uma captura realizada pelo servidor SA de um *ping* entre o próprio servidor e o portátil Bela. Como pode-se observar, houve a resposta aos *ping requests*. Também pode ser observado os *ARP requests e replies*, responsáveis por tornarem o *ping* exequível nesta situação fazendo com que ambas as máquinas tomassem conhecimento do MAC *address* uma da outra.

19	28.923447596	10.0.5.10	10.0.5.20	ICMP	98 Echo (ping) request	id=0x00ec, seq=1/256, ttl=64 (reply in 2..
20	28.924756256	10.0.5.20	10.0.5.10	ICMP	98 Echo (ping) reply	id=0x00ec, seq=1/256, ttl=64 (request in..
21	29.924765529	10.0.5.10	10.0.5.20	ICMP	98 Echo (ping) request	id=0x00ec, seq=2/512, ttl=64 (reply in 2..
22	29.925116495	10.0.5.20	10.0.5.10	ICMP	98 Echo (ping) reply	id=0x00ec, seq=2/512, ttl=64 (request in..
23	30.013264443	10.0.5.1	224.0.0.5	OSPF	78 Hello Packet	
24	30.928891954	10.0.5.10	10.0.5.20	ICMP	98 Echo (ping) request	id=0x00ec, seq=3/768, ttl=64 (reply in 2..
25	30.929161493	10.0.5.20	10.0.5.10	ICMP	98 Echo (ping) reply	id=0x00ec, seq=3/768, ttl=64 (request in..
26	31.953819236	10.0.5.10	10.0.5.20	ICMP	98 Echo (ping) request	id=0x00ec, seq=4/1024, ttl=64 (reply in ..
27	31.954488085	10.0.5.20	10.0.5.10	ICMP	98 Echo (ping) reply	id=0x00ec, seq=4/1024, ttl=64 (request i..
28	32.013727679	10.0.5.1	224.0.0.5	OSPF	78 Hello Packet	
29	32.976520288	10.0.5.10	10.0.5.20	ICMP	98 Echo (ping) request	id=0x00ec, seq=5/1280, ttl=64 (reply in ..
30	32.976709856	10.0.5.20	10.0.5.10	ICMP	98 Echo (ping) reply	id=0x00ec, seq=5/1280, ttl=64 (request i..
31	34.016605702	10.0.5.1	224.0.0.5	OSPF	78 Hello Packet	
33	34.096866723	00:00:00_aa:00:05	00:00:00_aa:00:04	ARP	42 Who has 10.0.5.20? Tell 10.0.5.10	
32	34.097406971	00:00:00_aa:00:04	00:00:00_aa:00:05	ARP	42 Who has 10.0.5.10? Tell 10.0.5.20	
34	34.097450925	00:00:00_aa:00:05	00:00:00_aa:00:04	ARP	42 10.0.5.10 is at 00:00:00_aa:00:05	
35	34.097520346	00:00:00_aa:00:04	00:00:00_aa:00:05	ARP	42 10.0.5.20 is at 00:00:00_aa:00:04	

Figura 41: Teste de conectividade IP - Bela & Servidor SA

D) Não volte a repor a rota por defeito. Adicione todas as rotas estáticas necessárias para restaurar a conectividade para o servidor SA, 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@Sa:/tmp/pycore.43093/Sa.conf# netstat -nr
Kernel IP routing table
Destination      Gateway         Genmask         Flags    MSS Window  irtt Iface
10.0.5.0          0.0.0.0         255.255.255.0   U        0 0        0 eth0
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf# sudo route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.5.1
root@Sa:/tmp/pycore.43093/Sa.conf#

```

Figura 42: Rotas adicionadas

Pode ser dito que, na ausência da rota por defeito, foi preciso introduzir o conhecimento completo de todas as redes da topologia. Nesta situação até que não foi muito trabalhoso, porém tal processo não é escalável. Conseguimos com isso entender a utilidade da rota por defeito.

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.

Como poderá ser visto na próxima página, serão mostrado três *ping's* que envolvem o servidor SA, porém tal comando voltou a ter sucesso entre qualquer máquina de qualquer uma das redes da topologia e o servidor SA. De seguida será mostrado também a nova tabela

de encaminhamento do servidor. Como poderá ser visto, com exceto da rede destino 10.0.5.0, todas as demais apresentam a *flag* G, indicando que ainda existirão intermediários no processo de *routing*.

```
root@Ariel:/tmp/pycore.43093/Ariel.conf# ping -c 5 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=1.54 ms
64 bytes from 10.0.5.10: icmp_seq=2 ttl=61 time=0.900 ms
64 bytes from 10.0.5.10: icmp_seq=3 ttl=61 time=0.837 ms
64 bytes from 10.0.5.10: icmp_seq=4 ttl=61 time=0.718 ms
64 bytes from 10.0.5.10: icmp_seq=5 ttl=61 time=0.493 ms

--- 10.0.5.10 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4014ms
rtt min/avg/max/mdev = 0.493/0.896/1.536/0.348 ms
root@Ariel:/tmp/pycore.43093/Ariel.conf#
```

Figura 43: Teste de conectividade IP - Ariel & Servidor SA

```
root@Sa:/tmp/pycore.43093/Sa.conf# ping -c 5 10.0.4.21
PING 10.0.4.21 (10.0.4.21) 56(84) bytes of data.
64 bytes from 10.0.4.21: icmp_seq=1 ttl=62 time=1.72 ms
64 bytes from 10.0.4.21: icmp_seq=2 ttl=62 time=0.689 ms
64 bytes from 10.0.4.21: icmp_seq=3 ttl=62 time=0.716 ms
64 bytes from 10.0.4.21: icmp_seq=4 ttl=62 time=0.702 ms
64 bytes from 10.0.4.21: icmp_seq=5 ttl=62 time=0.368 ms

--- 10.0.4.21 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4078ms
rtt min/avg/max/mdev = 0.368/0.838/1.719/0.458 ms
```

Figura 44: Teste de conectividade IP - Jasmine & Servidor SA

```
root@Sa:/tmp/pycore.43093/Sa.conf# ping -c 5 10.0.8.2
PING 10.0.8.2 (10.0.8.2) 56(84) bytes of data.
64 bytes from 10.0.8.2: icmp_seq=1 ttl=63 time=2.17 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=63 time=0.386 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=63 time=0.245 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=63 time=1.34 ms
64 bytes from 10.0.8.2: icmp_seq=5 ttl=63 time=0.305 ms

--- 10.0.8.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4038ms
rtt min/avg/max/mdev = 0.245/0.888/2.167/0.754 ms
```

Figura 45: Teste de conectividade IP - Router de acesso RISP & Servidor SA

```
root@Sa:/tmp/pycore.43093/Sa.conf# netstat -nr
Kernel IP routing table
Destination      Gateway         Genmask         Flags        MSS Window  irtt  Iface
10.0.0.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
10.0.1.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
10.0.2.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
10.0.3.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
10.0.4.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
10.0.5.0          0.0.0.0         255.255.255.0   U            0 0         0 eth0
10.0.6.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
10.0.7.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
10.0.8.0          10.0.5.1        255.255.255.0   UG           0 0         0 eth0
root@Sa:/tmp/pycore.43093/Sa.conf#
```

Figura 46: Nova tabela de encaminhamento servidor SA

3.3 Secção 3

A) Considere que dispõe apenas do endereço de rede IP 192.168.XXX.128/25, em que XXX é o decimal correspondendo ao seu número de grupo (PLXX). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo as redes de acesso externo e backbone inalteradas), sabendo que o número de departamentos pode vir a aumentar no curto prazo. Atribua endereços às interfaces dos vários sistemas envolvidos. Assuma que todos os endereços de sub-redes são usáveis. Justifique as opções tomadas no planeamento.

O endereço de rede IP em nosso contexto é 192.168.120.128/25. Como a máscara de rede é 255.255.255.128), significa que iremos ter cerca de 7 *bits* para definirmos nossas sub-redes e *estações*:

Máscara da rede IP

$$255.255.255.128 = 11111111.11111111.11111111.10000000$$

Deste modo, se separarmos apenas 2 *bits* para o espaço de endereçamento das sub-redes, iríamos restringir o número de departamentos a 4 (que é já a situação atual). Queremos então possibilitar a expansão de departamentos no curto prazo. Deste modo, optamos por reservar 3 *bits* para as sub-redes, o que vai ocasionar em 8 sub-redes no total (devido ao facto de todos os endereços de sub-redes serem usáveis). Deste modo, cada sub-rede poderá ter $2^4 - 2$ endereços IP's para distribuir entre as suas estações (a subtração por dois é oriunda do endereço reservado para *broadcast* e para o endereço da sub-rede. Ou seja, cada departamento poderá ter 14 *hosts*). Caso aumentássemos o número de *bits* para a sub-rede, iríamos limitar uma quantidade de 6 *hosts* por departamento, o que escolhemos não fazer por não parecer muito escalável.

Sub-Redes			
Sub-Rede	Primeiro IP válido	Último IP válido	Broadcast
192.168.120.128	192.168.120.129	192.168.120.142	192.168.120.143
192.168.120.144	192.168.120.145	192.168.120.158	192.168.120.159
192.168.120.160	192.168.120.161	192.168.177.174	192.168.120.175
192.168.120.176	192.168.120.177	192.168.177.190	192.168.120.191
192.168.120.192	192.168.120.193	192.168.177.206	192.168.120.207
192.168.120.208	192.168.120.209	192.168.177.222	192.168.120.223
192.168.120.224	192.168.120.225	192.168.177.238	192.168.120.239
192.168.120.240	192.168.120.241	192.168.177.254	192.168.120.255

Endereços atribuídos	
Equipamento (+ interface)	Endereço IP da interface
SA (eth0)	192.168.120.130/28
Monstro (eth0)	192.168.120.131/28
Bela (eth0)	192.168.120.129/28
Ra (eth0)	10.0.3.2/24
Ra (eth1)	10.0.0.2/24
Ra (eth3)	10.0.8.1/24
Ra (eth2)	192.168.120.132/28
Jasmine (eth0)	192.168.120.145/28
Alladin (eth0)	192.168.120.147/28
Rb (eth0)	192.168.120.146/28
Rb (eth0)	10.0.1.1/24
Rb (eth1)	10.0.0.1/24
Rb (eth2)	192.168.120.148/28
Eric (eth0)	192.168.120.163/28
Ariel (eth0)	192.168.120.161/28
Rc (eth0)	192.168.120.162/28
Rc (eth0)	10.0.1.2/24
Rc (eth1)	10.0.2.1/24
Rc (eth2)	192.168.120.164/28
Simba (eth0)	192.168.120.179/28
Nala (eth0)	192.168.120.177/28
Rd (eth0)	192.168.120.178/28
Rd (eth0)	10.0.3.1/24
Rd (eth1)	10.0.2.2/24
Rd (eth2)	192.168.120.180/28

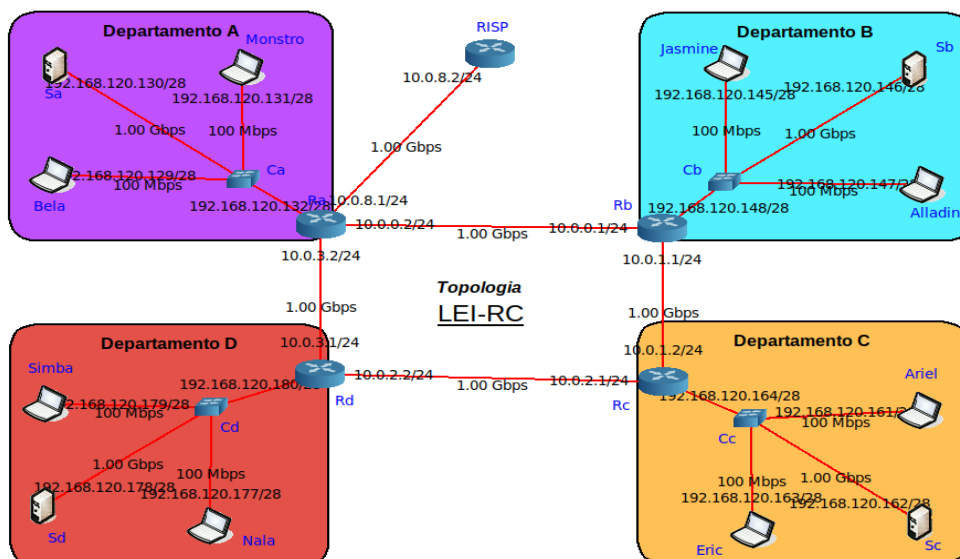


Figura 47: Topologia da rede desenvolvida

B) Qual a máscara de rede que usou (em formato decimal)? Quantos hosts IP pode interligar em cada departamento? Quantos prefixos de sub-rede ficam disponíveis para uso futuro? Justifique.

Máscara de rede

$255.255.255.240 = 11111111.11111111.11111111.11110000$

Podem interligar-se 14 *hosts* em cada departamento (de acordo com a conta mostrada na questão anterior). Ficam disponíveis 4 prefixos de sub-redes para o futuro (dado 4 já estarem sendo usado por departamento e ao todo serem 8 no total).

C) Verifique e garanta que a conectividade IP interna na rede local LEI-RC é mantida. No caso de não existência de conectividade, reveja a atribuição de endereços efetuada e eventuais erros de encaminhamento por forma a realizar as correções necessárias. Explique como procedeu.

Ao ser atualizada a topologia no *core* (de acordo com a tabela de endereços mostradas na primeira questão desta secção), nos deparamos com um problema interessante. Apesar dos *routers* dinamicamente atualizarem as suas tabelas de encaminhamento (pelos princípios citados na secção anterior), os *hosts* do sistema estavam considerando erroneamente o "próximo salto" a ser dado para a rota por defeito. Isto porque, como será mostrado no *print* a seguir, ao tentarmos realizar um *ping* do servidor SC para o servidor SA, tal comando não teve sucesso. A justificação para isto foi o incorreto valor atribuído pelo Core para o "próximo salto" na rota por defeito (sem utilizar protocolos dinâmicos), que estava sendo para o portátil Ariel (que foi inicialmente definido por nós com o endereço 192.168.120.161). Reparámos que o Core atribui como próximo salto da rota por defeito o primeiro IP da sub-rede. Deste modo, a conexão entre redes remotas torna-se impossível, pois apenas se o "próximo salto" fosse para o *router* do departamento que este saberia como lidar com pacotes entre redes remotas. Desta maneira poderíamos corrigir este problema de duas formas: retribuindo corretamente os endereços IP's por departamento de modo a tornar os *routers* como próximos saltos das rotas por defeito, ou alterar a tabela de encaminhamento dos próprios *end systems*. Procedemos de acordo com a segunda alternativa.

```
root@Sc:/tmp/pycore.43093/Sc.conf# netstat -nr
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt Iface
0.0.0.0          192.168.120.161 0.0.0.0         UG        0 0          0 eth0
192.168.120.160 0.0.0.0          255.255.255.240 U        0 0          0 eth0
root@Sc:/tmp/pycore.43093/Sc.conf#
```

Figura 48: Tabela de encaminhamento servidor SC

As rotas por defeito foram trocadas para os portáteis e servidores de cada departamento. Ou seja, o próximo salto da rota por defeito foi atualizado de modo a ser o *router* de cada

departamento. Segue um *print* com o procedimento realizado para o servidor Sc, porém há de se realçar que o mesmo processo foi feito para todos os *end systems*:

```
root@Sc:/tmp/pycore.41873/Sc.conf# netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags   MSS Window  irtt Iface
0.0.0.0            192.168.120.161   0.0.0.0           UG        0 0          0 eth0
192.168.120.160    0.0.0.0           255.255.255.240   U        0 0          0 eth0
root@Sc:/tmp/pycore.41873/Sc.conf# sudo route del -net 0.0.0.0 netmask 0.0.0.0 gw 192.168.120.161
root@Sc:/tmp/pycore.41873/Sc.conf# netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags   MSS Window  irtt Iface
192.168.120.160    0.0.0.0           255.255.255.240   U        0 0          0 eth0
root@Sc:/tmp/pycore.41873/Sc.conf# sudo route add -net 0.0.0.0 netmask 0.0.0.0 gw 192.168.120.164
root@Sc:/tmp/pycore.41873/Sc.conf# netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags   MSS Window  irtt Iface
0.0.0.0            192.168.120.164   0.0.0.0           UG        0 0          0 eth0
192.168.120.160    0.0.0.0           255.255.255.240   U        0 0          0 eth0
root@Sc:/tmp/pycore.41873/Sc.conf#
```

Figura 49: Troca da rota por defeito

Por fim foi realizado alguns *ping's* entre departamentos que tiveram sucesso de resposta:

```
root@Sc:/tmp/pycore.41873/Sc.conf# ping -c 5 192.168.120.130
PING 192.168.120.130 (192.168.120.130) 56(84) bytes of data:
64 bytes from 192.168.120.130: icmp_seq=1 ttl=61 time=3.86 ms
64 bytes from 192.168.120.130: icmp_seq=2 ttl=61 time=0.665 ms
64 bytes from 192.168.120.130: icmp_seq=3 ttl=61 time=0.542 ms
64 bytes from 192.168.120.130: icmp_seq=4 ttl=61 time=0.565 ms
64 bytes from 192.168.120.130: icmp_seq=5 ttl=61 time=1.09 ms

--- 192.168.120.130 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4063ms
rtt min/avg/max/mdev = 0.542/1.345/3.863/1.274 ms
root@Sc:/tmp/pycore.41873/Sc.conf#
```

Figura 50: Ping entre o servidor Sc e Sa

A imagem que se segue mostra também a tabela de encaminhamento feito dinamicamente pelo *router* Rb que confirma as sub-redes criadas, assim como a máscara explicada nas questões anteriores:

```
root@Rb:/tmp/pycore.39059/Rb.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 eth1
10.0.1.0            0.0.0.0           255.255.255.0     U        0 0          0 eth0
10.0.2.0            10.0.1.2          255.255.255.0     UG       0 0          0 eth0
10.0.3.0            10.0.0.2          255.255.255.0     UG       0 0          0 eth1
10.0.8.0            10.0.0.2          255.255.255.0     UG       0 0          0 eth1
192.168.120.128     10.0.0.2          255.255.255.240   UG       0 0          0 eth1
192.168.120.144     0.0.0.0           255.255.255.240   U        0 0          0 eth2
192.168.120.160     10.0.1.2          255.255.255.240   UG       0 0          0 eth0
192.168.120.176     10.0.1.2          255.255.255.240   UG       0 0          0 eth0
root@Rb:/tmp/pycore.39059/Rb.conf#
```

Figura 51: Tabela encaminhamento *router* Rb

4 Conclusão

Este trabalho contribuiu de forma notável para o desenvolvimento do conhecimento do nosso grupo a respeito do protocolo IPv4. Na segunda parte, nomeadamente sobre o funcionamento das **tabelas de encaminhamento**, optamos por dar um maior cuidado no desenvolvimento das respostas. Este cuidado de mostrar o raciocínio completo por trás da busca pelas respostas ajudou-nos a compreender grande parte do processo envolvido na transmissão de um pacote de informação de um sistema a outro a nível de rede. Com isso acabamos por nos deparar com alguns conceitos para além do que era esperado para o trabalho, como por exemplo a análise feita sobre o protocolo ARP e OSPF. Acreditamos termos alcançado um nível detalhado da explicação sobre as consequências de ocorrer uma variação nas tabelas de encaminhamento de algum sistema de uma rede, assim como a interpretação feita sobre a estrutura e informações a serem extraídas de tais tabelas.

Para além da *Secção 2* da segunda parte do trabalho, foi mostrado também um novo esquema de endereçamento para as redes dos departamentos mediante as condições propostas no enunciado. Este novo esquema foi devidamente descrito e a sua adaptação prática a partir da teoria desenvolvida ocorreu de acordo com o planeado, sem levantar muitas dificuldades. Sabíamos da secção anterior que os novos esquemas de endereçamento podiam levantar alguns erros iniciais, principalmente pelos nossos actos de alterarmos algumas configurações (atribuição de endereços IP's) previamente definidos pelo Core. Este erro foi encontrado e residia sobre as tabelas de encaminhamento estaticamente definidas pelo Core estarem inconsistentes devido a rota *default* estar mal configurada. Deste modo foi proposta uma das possíveis resoluções, concertando o fluxo entre as redes dos diversos departamentos. Este processo foi devidamente documentado.

Em suma, todas as secções abordadas no trabalho enriqueceram nosso aprendizado sobre a camada de rede. A própria parte 1 ajudou-nos a observar resultados práticos e confirmar a teoria por trás da *feature* de fragmentação envolvida no protocolo IPv4. Praticamente passamos por todos os campos do cabeçalho para podermos construir nossas respostas. Dito isto, este relatório apresenta respostas descritivas a respeito de muitas questões levantadas por um protocolo tão importante para o funcionamento de nosso mundo digital atual.