

Protocolo IPv4

Etienne Costa(A76089) Joana Cruz(A76270)
Rafael Alves(A72629)

Resumo

O principal objectivo deste trabalho é o estudo do Internet Protocol (IP) nas principais vertentes, nomeadamente: estudo do formato de um pacote ou datagrama IP, fragmentação de pacotes IP, endereçamento IP e encaminhamento IP.

1 Parte I

1.1 Exercício 1

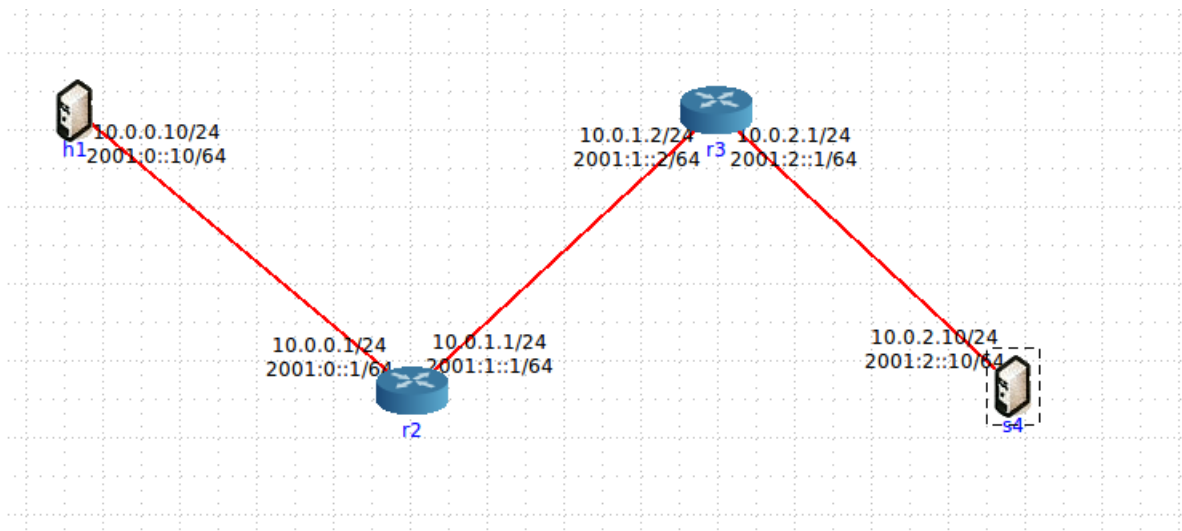


Figura 1: Topologia Core.

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

Solution:

```
root@h1:/tmp/pycore.48531/h1.conf# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1  A0 (10.0.0.1)  0.059 ms  0.006 ms  0.005 ms
 2  10.0.1.2 (10.0.1.2)  0.031 ms  0.009 ms  0.008 ms
 3  10.0.2.10 (10.0.2.10)  0.029 ms  0.011 ms  0.010 ms
```

Figura 2: Resultado do Traceroute

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

Solution: Inicialmente, o host h1 envia 3 pacotes com o TTL igual a 1 que por sua vez são descartados pelo router r2 pois excedia o TTL. De seguida, o TTL é incrementado para 2 e são enviados 3 pacotes que são descartados pelo router r3, pois também excedia o TTL. Finalmente, o TTL é incrementado para 3 fazendo com que os pacotes cheguem até ao host s4, devolvendo uma mensagem (Echo ping) reply.

102	468.456487	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=1/256, ttl=1
103	468.456502	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
104	468.456510	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=2/512, ttl=1
105	468.456514	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
106	468.456517	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=3/768, ttl=1
107	468.456521	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
108	468.456524	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=4/1024, ttl=2
109	468.456554	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
110	468.456558	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=5/1280, ttl=2
111	468.456565	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
112	468.456568	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=6/1536, ttl=2
113	468.456574	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)	
114	468.456578	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=7/1792, ttl=3
115	468.456605	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=7/1792, ttl=62
116	468.456610	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=8/2048, ttl=3
117	468.456619	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=8/2048, ttl=62
118	468.456623	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=9/2304, ttl=3
119	468.456632	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=9/2304, ttl=62
120	468.456635	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=10/2560, ttl=4
121	468.456644	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=10/2560, ttl=62
122	468.456647	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=11/2816, ttl=4
123	468.456656	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=11/2816, ttl=62
124	468.456659	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=12/3072, ttl=4
125	468.456668	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=12/3072, ttl=62
126	468.456672	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=13/3328, ttl=5
127	468.456681	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=13/3328, ttl=62
128	468.456684	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=14/3584, ttl=5
129	468.456693	10.0.2.10	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x002f, seq=14/3584, ttl=62
130	468.456696	10.0.0.10	10.0.2.10	ICMP	74 Echo (ping) request	id=0x002f, seq=15/3840, ttl=5

Figura 3: Tráfego ICMP

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

Solution: O valor mínimo que o campo TTL deve ter para alcançar o destino s4 é 3.

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

Solution: O valor médio do tempo de ida e volta é 0.017 ms.

$$RoundTripTime = \frac{(0.029ms + 0.011ms + 0.010ms)}{3} \quad (1)$$

1.2 Exercício 2

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

```

[~]-[etiennecesta@MacBook-Air-de-Etienne]-[0]-[6742]
[(:)] % traceroute -I marco.uminho.pt
traceroute to marco.uminho.pt (193.136.9.240), 64 hops max, 72 byte packets
 1 172.26.254.254 (172.26.254.254)  2.121 ms  1.500 ms  1.600 ms
 2 172.16.2.1 (172.16.2.1)  1.311 ms  3.606 ms  1.807 ms
 3 172.16.115.252 (172.16.115.252)  1.673 ms  3.992 ms  1.766 ms
 4 marco.uminho.pt (193.136.9.240)  1.688 ms  1.646 ms  1.589 ms
[~]-[etiennecesta@MacBook-Air-de-Etienne]-[0]-[6743]
[(:)] % ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
    inet 127.0.0.1 netmask 0xffff0000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
    nd6 options=201<PERFORMNUD,DAD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
EHC250: flags=0<> mtu 0
EHC253: flags=0<> mtu 0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    ether 10:40:f3:92:98:20
    inet6 fe80::402:937e:c941:6a50%en0 prefixlen 64 secured scopeid 0x6
    inet 172.26.64.93 netmask 0xffff0000 broadcast 172.26.255.255
    nd6 options=201<PERFORMNUD,DAD>

```

Figura 4: Resultado do ifconfig.

Figura 4: Resultado do ifconfig.

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

Solution: O valor do campo protocolo é o ICMP ou seja Internet Control Message Protocol, que é um protocolo utilizado para fornecer relatórios de erros à fonte original.

3. Quantos bytes tem o cabeçalho IP(v4)? Quantos bytes tem o campo de dados (payload) do datagrama? Como se calcula o tamanho do payload?

Solution: O cabeçalho IP possui 20 bytes. O campo de dados do datagrama possui 52 bytes e é calculado através da fórmula abaixo indicada.

$$Payload = TotalLength - HeaderLength \quad (2)$$

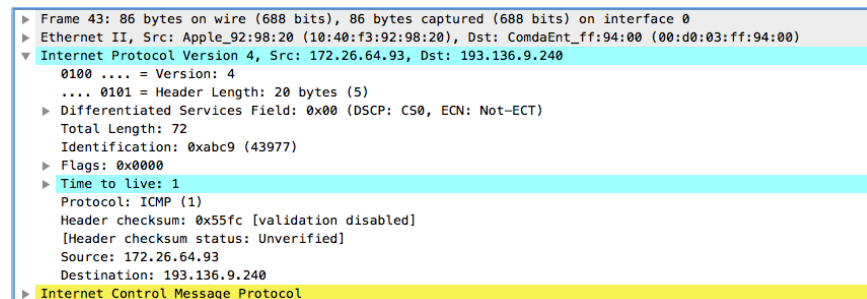


Figura 5: Primeira mensagem ICMP.

4. O datagrama IP foi fragmentado? Justifique.

Solution: Não, pois as flags Fragment Offset e More Fragments são iguais a 0.

5. 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.

Solution: No cabeçalho IP os campos que variam são: Identification e o Header Checksum.

```
► Frame 43: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
► Ethernet II, Src: Apple_92:98:20 (10:40:f3:92:98:20), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.64.93, 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: 72
    Identification: 0xabc9 (43977)
    ► Flags: 0x0000
    ► Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x55fc [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.64.93
    Destination: 193.136.9.240
► Internet Control Message Protocol
```

Figura 6: Primeiro Pacote.

```
► Frame 47: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
► Ethernet II, Src: Apple_92:98:20 (10:40:f3:92:98:20), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.64.93, 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: 72
    Identification: 0xabca (43978)
    ► Flags: 0x0000
    ► Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x55fb [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.64.93
    Destination: 193.136.9.240
► Internet Control Message Protocol
```

Figura 7: Segundo Pacote.

```
► Frame 49: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface 0
► Ethernet II, Src: Apple_92:98:20 (10:40:f3:92:98:20), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.64.93, 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: 72
    Identification: 0xabc9 (43979)
    ► Flags: 0x0000
    ► Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x55fa [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.64.93
    Destination: 193.136.9.240
► Internet Control Message Protocol
```

Figura 8: Terceiro Pacote.

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

Solution: Sim. Os valores do campo de Identificação são incrementados em uma unidade por cada pacote enviado e o TTL é incrementado uma unidade a cada 3 pacotes.

7. 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?

Solution: O valor do campo TTL é 255, e o mesmo não permanece constante visto que é decrementado em 1 unidade a cada 3 pacotes.

1.3 Exercício 3

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

Solution: Houve a necessidade de fragmentar o pacote inicial pois o mesmo excede o Maximum Transmission Unit. O Maximum Transmission Unit refere-se ao tamanho do maior pacote que uma camada de um protocolo de comunicação pode transmitir.

2. 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?

Solution: As informações do datagrama IP que indicam que o pacote foi fragmentado são os respectivos valores que as flags Fragment Offset e More Fragments tomam, sendo que podemos afirmar que se trata do primeiro fragmento pois as mesmas tomam os valores 0 e 1. O tamanho do datagrama é de 3535 bytes.

```
► Frame 166: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
► Ethernet II, Src: Apple_92:98:20 (10:40:f3:92:98:20), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.64.93, 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: 0xb94f (47439)
    ▼ Flags: 0x2000, More fragments
        0... .. = Reserved bit: Not set
        .0.. .. = Don't fragment: Not set
        ..1. .. = More fragments: Set
        ...0 0000 0000 0000 = Fragment offset: 0
    ► Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x22e2 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.64.93
    Destination: 193.136.9.240
    Reassembled IPv4 in frame: 168
    ► Data (1480 bytes)
```

Figura 9: Primeiro Fragmento.

3. Imprima o segundo fragmento do datagrama IP original. Que informação do cabeçalho IP indica que não se trata do primeiro fragmento? Há mais fragmentos? O que nos permite afirmar isso?

Solution: Sabe-se que não se trata do primeiro fragmento devido ao valor da flag Fragment Offset , sendo que podemos afirmar que não se trata do primeiro fragmento pois a mesma toma um valor diferente de 0. Podemos afirmar que existem mais fragmentos pois a flag More Fragments tem o valor 1.

```
► Frame 167: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
► Ethernet II, Src: Apple_92:98:20 (10:40:f3:92:98:20), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.64.93, 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: 0xb94f (47439)
    ▼ Flags: 0x20b9, More fragments
        0... .... = Reserved bit: Not set
        .0. .... = Don't fragment: Not set
        ..1. .... = More fragments: Set
        ...0 0000 1011 1001 = Fragment offset: 185
    ► Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x2229 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.64.93
    Destination: 193.136.9.240
    Reassembled IPv4 in frame: 168
    ► Data (1480 bytes)
```

Figura 10: Segundo Fragmento.

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

Solution: A partir do datagrama original foram criados 3 fragmentos, tendo cada um o seguinte tamanho:

- **Primeiro Fragmento:** 1480 bytes.
- **Segundo Fragmento:** 1480 bytes.
- **Último Fragmento:** 575 bytes.

Podemos ainda detectar o último fragmento através das flags Fragment Offset e More Fragments, pois o Fragment Offset terá um valor diferente de 0 e a More Fragments será igual a 0, podendo ainda fazer-se a verificação de que se trata do mesmo datagrama através do campo Identification.

```

▶ Frame 168: 609 bytes on wire (4872 bits), 609 bytes captured (4872 bits) on interface 0
▶ Ethernet II, Src: Apple_92:98:20 (10:40:f3:92:98:20), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
▼ Internet Protocol Version 4, Src: 172.26.64.93, 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: 595
        Identification: 0xb94f (47439)
    ▼ Flags: 0x0172
        0... .. = Reserved bit: Not set
        ..0. .... = Don't fragment: Not set
        ...0. .... = More fragments: Not set
        ...0 0001 0111 0010 = Fragment offset: 370
    ▶ Time to live: 1
        Protocol: ICMP (1)
        Header checksum: 0x44f9 [validation disabled]
        [Header checksum status: Unverified]
        Source: 172.26.64.93
        Destination: 193.136.9.240
    ▶ [3 IPv4 Fragments (3535 bytes): #166(1480), #167(1480), #168(575)]
    ▶ Internet Control Message Protocol

```

Figura 11: Último Fragmento.

5. 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.

Solution: Os campos que mudam no cabeçalho entre os diferentes fragmentos são:

- **Flag More Fragments.**
- **Flag Fragment Offset.**
- **Header Cheksum.**

Um receptor sabe que um pacote é um fragmento caso a sua flag More Fragments esteja activa (excepto no último fragmento) e caso a flag Fragment Offset do Fragmento tenha um valor diferente de 0 (excepto para o primeiro fragmento). Os fragmentos que possuírem a mesma identificação pertencem ao mesmo datagrama, e o campo do Offset do fragmento permite ordenar esses fragmentos. Ao receber o último fragmento o receptor pode então calcular o tamanho do campo de dados tirando partido da seguinte fórmula:

$$TamanhoTotal = Offset * 8 + (TotalLength - HeaderLength) \quad (3)$$

Sendo que o Offset e o Total Length correspondem a valores do último fragmento. Logo o pacote é remontado no seu destino final e é enviado para a camada superior concretamente a de Transporte.

2 Parte II

2.1 Exercício 1

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

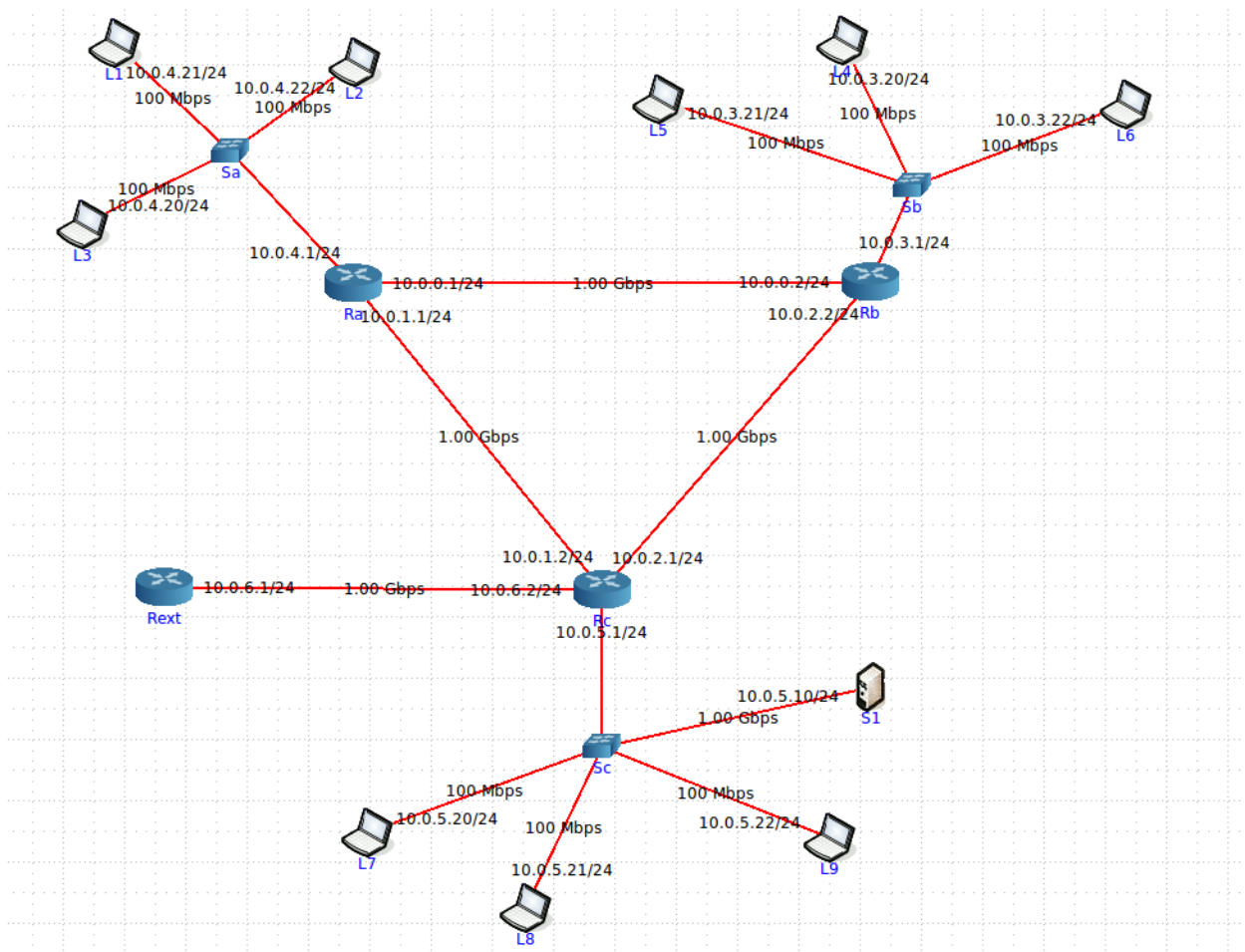


Figura 12: Endereços IP e máscaras de rede atribuídas pelo CORE.

2. Trata-se de endereços públicos ou privados? Porquê?

Solution: Tratam-se de endereços privados pois pertencem à gama de valores da classe A: de 10.0.0.0 a 10.255.255.255 (10.0.0.0 /8).

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

Solution: Porque os switches trabalham na camada de dados e não têm capacidade para trabalhar na cama de rede IP.

4. Certifique-se que existe conectividade IP entre os laptops dos vários departamentos e o servidor do departamento C.

Solution:


```
vcmd
root@L1:/tmp/pycore.48538/L1.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_req=1 ttl=62 time=0.145 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=62 time=0.075 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=62 time=0.073 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=62 time=0.152 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=62 time=0.080 ms
64 bytes from 10.0.5.10: icmp_req=6 ttl=62 time=0.071 ms
64 bytes from 10.0.5.10: icmp_req=7 ttl=62 time=0.078 ms
64 bytes from 10.0.5.10: icmp_req=8 ttl=62 time=0.213 ms
64 bytes from 10.0.5.10: icmp_req=9 ttl=62 time=0.077 ms
64 bytes from 10.0.5.10: icmp_req=10 ttl=62 time=0.078 ms
64 bytes from 10.0.5.10: icmp_req=11 ttl=62 time=0.093 ms
64 bytes from 10.0.5.10: icmp_req=12 ttl=62 time=0.078 ms
64 bytes from 10.0.5.10: icmp_req=13 ttl=62 time=0.079 ms
64 bytes from 10.0.5.10: icmp_req=14 ttl=62 time=0.184 ms
64 bytes from 10.0.5.10: icmp_req=15 ttl=62 time=0.077 ms
^C
--- 10.0.5.10 ping statistics ---
15 packets transmitted, 15 received, 0% packet loss, time 1399ms
rtt min/avg/max/mdev = 0.071/0.103/0.213/0.046 ms
root@L1:/tmp/pycore.48538/L1.conf#
```

Figura 13: Conetividade IP entre o laptop L1 e servidor S1.

```
vcmd
root@L4:/tmp/pycore.48538/L4.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_req=1 ttl=62 time=0.117 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=62 time=0.081 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=62 time=0.201 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=62 time=0.075 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=62 time=0.077 ms
64 bytes from 10.0.5.10: icmp_req=6 ttl=62 time=0.232 ms
64 bytes from 10.0.5.10: icmp_req=7 ttl=62 time=0.082 ms
^C
--- 10.0.5.10 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 599ms
rtt min/avg/max/mdev = 0.075/0.123/0.232/0.061 ms
root@L4:/tmp/pycore.48538/L4.conf# ^C
root@L4:/tmp/pycore.48538/L4.conf#
```

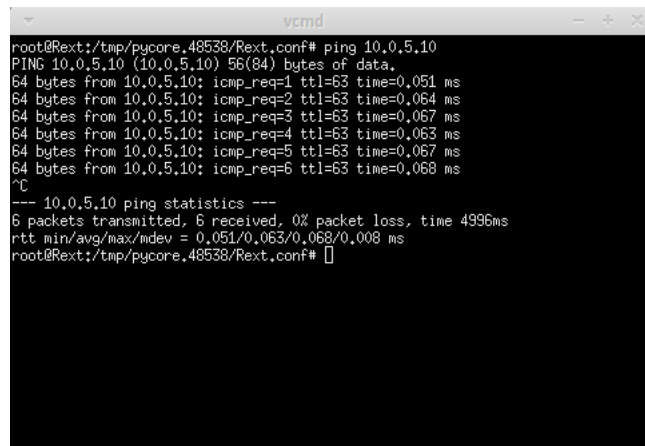
Figura 14: Conetividade IP entre o laptop L4 e servidor S1.

```
vcmd
root@L7:/tmp/pycore.48538/L7.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_req=1 ttl=64 time=0.096 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=64 time=0.047 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=64 time=0.047 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=64 time=0.054 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=64 time=0.053 ms
64 bytes from 10.0.5.10: icmp_req=6 ttl=64 time=0.052 ms
64 bytes from 10.0.5.10: icmp_req=7 ttl=64 time=0.038 ms
^C
--- 10.0.5.10 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 599ms
rtt min/avg/max/mdev = 0.038/0.055/0.096/0.018 ms
root@L7:/tmp/pycore.48538/L7.conf#
```

Figura 15: Conetividade IP entre o laptop L7 e servidor S1.

5. Certifique-se que existe conectividade IP do router de acesso Rext para o servidor S1.

Solution:



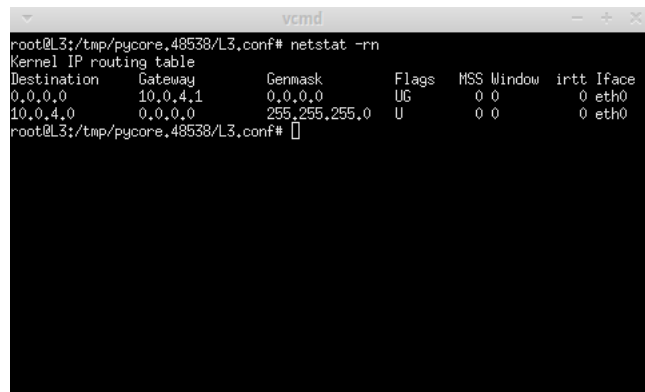
```
vcmd
root@Rext:/tmp/pycore.48538/Rext.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_req=1 ttl=63 time=0.051 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=63 time=0.064 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=63 time=0.067 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=63 time=0.063 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=63 time=0.067 ms
64 bytes from 10.0.5.10: icmp_req=6 ttl=63 time=0.068 ms
^C
--- 10.0.5.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4996ms
rtt min/avg/max/mdev = 0.051/0.063/0.068/0.008 ms
root@Rext:/tmp/pycore.48538/Rext.conf#
```

Figura 16: Conetividade IP entre Rext e servidor S1.

2.2 Exercício 2

1. Execute o comando netstat -rn por forma a poder consultar a tabela de encaminhamento unicast(IPv4).

Solution:



```
vcmd
root@L3:/tmp/pycore.48538/L3.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.4.1 0.0.0.0 UG 0 0 0 eth0
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@L3:/tmp/pycore.48538/L3.conf#
```

Figura 17: Tabela de encaminhamento do laptop L3 do departamento A.

```

root@Ra:/tmp/pycore.48538/Ra.conf# netstat -rn
Kernel IP routing table
Destination        Gateway            Genmask           Flags   MSS Window  irtt Iface
10.0.0.0            0.0.0.0            255.255.255.0     U        0 0          0 eth0
10.0.1.0            0.0.0.0            255.255.255.0     U        0 0          0 eth1
10.0.2.0            10.0.0.2           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 eth0
10.0.4.0            0.0.0.0            255.255.255.0     U        0 0          0 eth2
10.0.5.0            10.0.1.2           255.255.255.0     UG       0 0          0 eth1
10.0.6.0            10.0.1.2           255.255.255.0     UG       0 0          0 eth1
root@Ra:/tmp/pycore.48538/Ra.conf# 

```

Figura 18: Tabela de encaminhamento do router do departamento A.

2. Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico.

Solution: Laptop: Tirando partido do comando `ps -A`, podemos verificar que não está a correr nenhum protocolo de encaminhamento logo o encaminhamento é estático.

```

root@L3:/tmp/pycore.48539/L3.conf# ps -A
PID TTY          TIME CMD
1 ?            00:00:00 vncd
46 pts/3      00:00:00 bash
57 pts/3      00:00:00 ps
root@L3:/tmp/pycore.48539/L3.conf# 

```

Figura 19: Processos que correm no laptop L3.

Router: Tirando partido do comando `ps -A`, podemos verificar que está a correr o protocolo `ospfd` logo o encaminhamento é dinâmico.

```

root@Ra:/tmp/pycore.48539/Ra.conf# ps -A
PID TTY          TIME CMD
1 ?            00:00:00 vncd
41 ?            00:00:00 zebra
51 ?            00:00:00 ospf6d
71 ?            00:00:00 ospfd
90 pts/8       00:00:00 bash
101 pts/8      00:00:00 ps
root@Ra:/tmp/pycore.48539/Ra.conf# 

```

Figura 20: Processos que correm no router do departamento A.

3. Admita que, por questões administrativas, a rota por defeito (0.0.0.0 ou default) deve ser retirada definitivamente da tabela de encaminhamento do servidor S1 localizado no departamento C. Use o comando `route delete` para o efeito. Que implicações tem esta medida para os utilizadores da empresa que acedem ao servidor. Justifique.

Solution: Com a remoção da rota por defeito não seria possível enviar ou receber qualquer datagrama no servidor.

```

vcmd
root@S1:/tmp/pycore,48539/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0,0,0,0 10,0,5,1 0,0,0,0 UG 0 0 0 eth0
10,0,5,0 0,0,0,0 255,255,255,0 U 0 0 0 eth0
root@S1:/tmp/pycore,48539/S1.conf# route delete default
root@S1:/tmp/pycore,48539/S1.conf# netstat -rn
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@S1:/tmp/pycore,48539/S1.conf# 

```

Figura 21: Remoção da rota por defeito.

4. Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor S1. Utilize para o efeito o comando route add e registre os comandos que usou.

Solution:

```

vcmd
root@S1:/tmp/pycore,48539/S1.conf# netstat -rn
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@S1:/tmp/pycore,48539/S1.conf# route add default gw 10,0,5,1
root@S1:/tmp/pycore,48539/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0,0,0,0 10,0,5,1 0,0,0,0 UG 0 0 0 eth0
10,0,5,0 0,0,0,0 255,255,255,0 U 0 0 0 eth0
root@S1:/tmp/pycore,48539/S1.conf# 

```

Figura 22: Adição da rota estática.

5. Teste a nova política de encaminhamento garantindo que o servidor está novamente acessível, utilizando para o efeito o comando ping.

Solution:

```
vcmd
root@L3:/tmp/pycore.48539/L3.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_req=1 ttl=62 time=0.151 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=62 time=0.081 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=62 time=0.081 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=62 time=0.001 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=62 time=0.058 ms
64 bytes from 10.0.5.10: icmp_req=6 ttl=62 time=0.124 ms
64 bytes from 10.0.5.10: icmp_req=7 ttl=62 time=0.198 ms
64 bytes from 10.0.5.10: icmp_req=8 ttl=62 time=0.074 ms
^C
--- 10.0.5.10 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 6998ms
rtt min/avg/max/mdev = 0.001/0.096/0.198/0.056 ms
root@L3:/tmp/pycore.48539/L3.conf# ^C
root@L3:/tmp/pycore.48539/L3.conf#
```

Figura 23: Conetividade IP entre laptop L3 e servidor S1.

```
vcmd
root@Ra:/tmp/pycore.48539/Ra.conf# ping 10.0.5.10
PING 10.0.5.10 (10.0.5.10) 56(84) bytes of data:
64 bytes from 10.0.5.10: icmp_req=1 ttl=63 time=0.063 ms
64 bytes from 10.0.5.10: icmp_req=2 ttl=63 time=0.062 ms
64 bytes from 10.0.5.10: icmp_req=3 ttl=63 time=0.063 ms
64 bytes from 10.0.5.10: icmp_req=4 ttl=63 time=0.663 ms
64 bytes from 10.0.5.10: icmp_req=5 ttl=63 time=0.067 ms
64 bytes from 10.0.5.10: icmp_req=6 ttl=63 time=0.070 ms
64 bytes from 10.0.5.10: icmp_req=7 ttl=63 time=0.063 ms
64 bytes from 10.0.5.10: icmp_req=8 ttl=63 time=0.067 ms
64 bytes from 10.0.5.10: icmp_req=9 ttl=63 time=0.065 ms
64 bytes from 10.0.5.10: icmp_req=10 ttl=63 time=0.068 ms
64 bytes from 10.0.5.10: icmp_req=11 ttl=63 time=0.069 ms
^C
--- 10.0.5.10 ping statistics ---
11 packets transmitted, 11 received, 0% packet loss, time 9999ms
rtt min/avg/max/mdev = 0.062/0.120/0.663/0.171 ms
root@Ra:/tmp/pycore.48539/Ra.conf# ^C
root@Ra:/tmp/pycore.48539/Ra.conf#
```

Figura 24: Conetividade IP entre Ra e servidor S1.

2.3 Exercício 3

1. Considere que dispõe apenas do endereço de rede IP 172.XX.48.0/20, 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.

Solution: Sendo que a nossa topologia possui 3 departamentos para fazer o sub-netting seria necessário pelo menos 2 bits para as subredes, mas visto que duas das quatro opções correspondem a endereços privados só seria possível fazer o subnetting a dois dos três departamentos. Trabalhando com 3 bits teríamos disponível 6 opções válidas para definir as subredes. Logo teríamos 9 bits reservados para os hosts e 3 bits para as subredes.

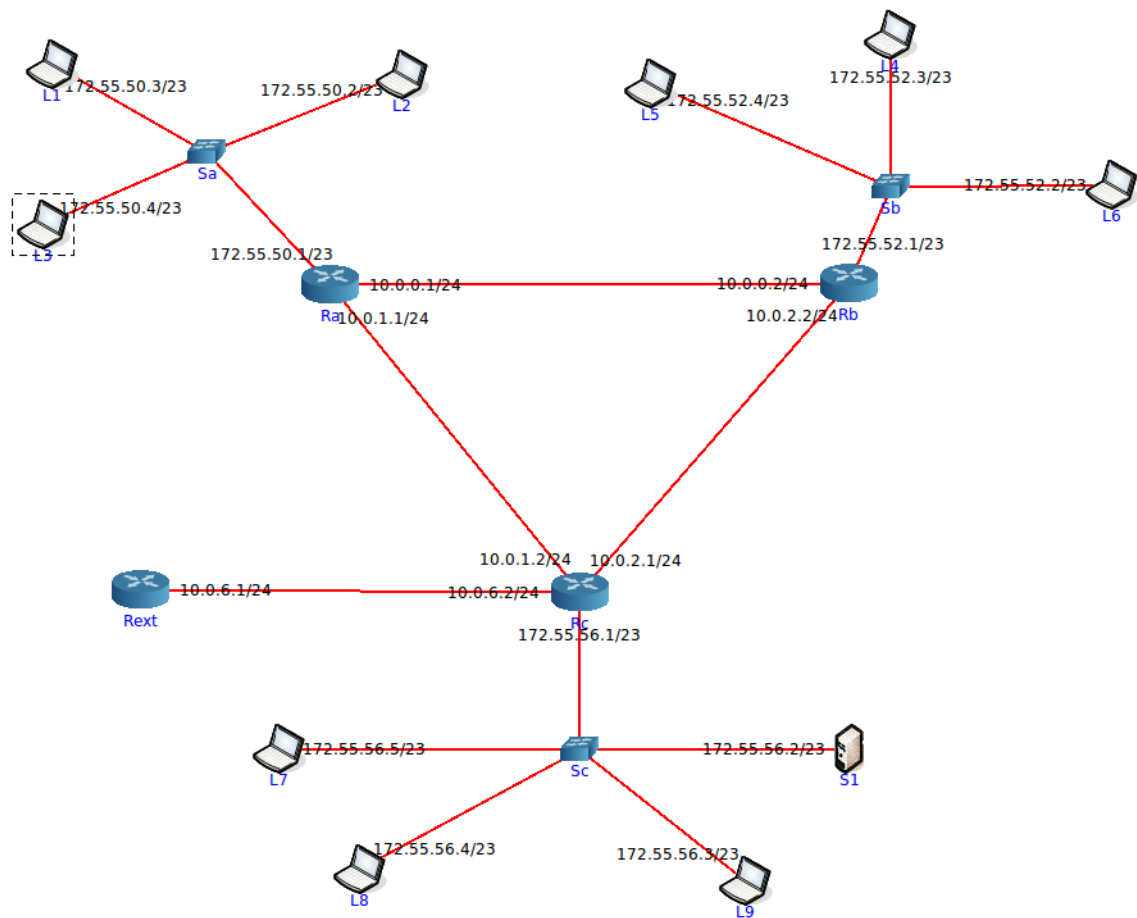


Figura 25: Topologia após o subnetting.

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

Solution:

Mascara : 255.255.254.0 (4)

Hosts : $2^9 - 2 = 510$ (5)

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

Solution:

```
vcmd
root@L3:/tmp/pycore.48540/L3.conf# ping 172.55.56.5
PING 172.55.56.5 (172.55.56.5) 56(84) bytes of data:
64 bytes from 172.55.56.5: icmp_req=1 ttl=62 time=0.141 ms
64 bytes from 172.55.56.5: icmp_req=2 ttl=62 time=0.072 ms
64 bytes from 172.55.56.5: icmp_req=3 ttl=62 time=0.079 ms
64 bytes from 172.55.56.5: icmp_req=4 ttl=62 time=0.075 ms
64 bytes from 172.55.56.5: icmp_req=5 ttl=62 time=0.079 ms
64 bytes from 172.55.56.5: icmp_req=6 ttl=62 time=0.046 ms
64 bytes from 172.55.56.5: icmp_req=7 ttl=62 time=0.080 ms
^C
--- 172.55.56.5 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6000ms
rtt min/avg/max/mdev = 0.046/0.081/0.141/0.028 ms
root@L3:/tmp/pycore.48540/L3.conf# ^C
root@L3:/tmp/pycore.48540/L3.conf#
```

Figura 26: Conetividade IP entre departamento A e B.

```
vcmd
root@L3:/tmp/pycore.48540/L3.conf# ping 172.55.52.3
PING 172.55.52.3 (172.55.52.3) 56(84) bytes of data:
From 172.55.50.1 icmp_seq=1 Destination Net Unreachable
From 172.55.50.1 icmp_seq=2 Destination Net Unreachable
From 172.55.50.1 icmp_seq=3 Destination Net Unreachable
From 172.55.50.1 icmp_seq=4 Destination Net Unreachable
From 172.55.50.1 icmp_seq=5 Destination Net Unreachable
From 172.55.50.1 icmp_seq=6 Destination Net Unreachable
From 172.55.50.1 icmp_seq=7 Destination Net Unreachable
From 172.55.50.1 icmp_seq=8 Destination Net Unreachable
^C
--- 172.55.52.3 ping statistics ---
8 packets transmitted, 0 received, +8 errors, 100% packet loss, time 6998ms
root@L3:/tmp/pycore.48540/L3.conf#
```

Figura 27: Conetividade IP entre departamento A e C.

```
vcmd
root@L4:/tmp/pycore.48540/L4.conf# ping 172.55.56.4
PING 172.55.56.4 (172.55.56.4) 56(84) bytes of data:
64 bytes from 172.55.56.4: icmp_req=1 ttl=62 time=0.138 ms
64 bytes from 172.55.56.4: icmp_req=2 ttl=62 time=0.081 ms
64 bytes from 172.55.56.4: icmp_req=3 ttl=62 time=0.081 ms
64 bytes from 172.55.56.4: icmp_req=4 ttl=62 time=0.080 ms
64 bytes from 172.55.56.4: icmp_req=5 ttl=62 time=0.227 ms
64 bytes from 172.55.56.4: icmp_req=6 ttl=62 time=0.076 ms
64 bytes from 172.55.56.4: icmp_req=7 ttl=62 time=0.059 ms
^C
--- 172.55.56.4 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 5998ms
rtt min/avg/max/mdev = 0.059/0.106/0.227/0.054 ms
root@L4:/tmp/pycore.48540/L4.conf# ^C
root@L4:/tmp/pycore.48540/L4.conf#
```

Figura 28: Conetividade IP entre departamento B e C.

Tirando partido do comando ping conseguimos ver que existe conectividade entre as várias redes locais.

3 Conclusão

Quanto à primeira parte do TP podemos concluir que:

- No envio de datagramas, por forma a impedir que estes sejam reencaminhados indefinidamente, é usado um parâmetro presente no cabeçalho IP designado TTL, que limita o número de *hop's*. Desta forma, para que um datagrama seja entregue num destino a N *hop's* de distância, é necessário que o TTL tenha um valor mínimo de N.
- Quando o tamanho dos datagramas excede a MTU é necessário recorrer a um processo designado de fragmentação que consiste em dividir os datagramas em partes mais pequenas(fragmentos) que são reassemblados no destino recorrendo a flags como: *Fragment Offset* e *More Fragments*.

Relativamente à segunda parte do TP, concluímos que os *hosts/end systems* não mantêm tabelas de encaminhamento extensas, visto que a maioria do tráfego é encaminhado para o *router*, sendo que estes últimos recorrem a protocolos de roteamento(OSPF, RIP, etc) para determinar as melhores rotas para um determinado destino. Por outro lado, e por forma a limitar o tamanho das tabelas, o encaminhamento é feito salto a salto i.e. o *router* apenas se preocupa com o próximo *hop*.