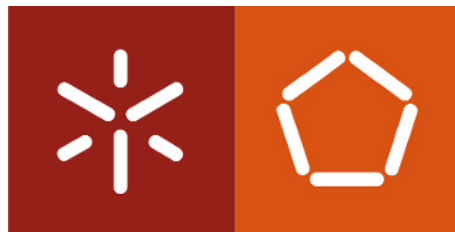


Redes de Computadores

11 de Novembro de 2019

Grupo nr. 15

a83899	André Moraes
a85954	Luís Ribeiro
a84783	Pedro Rodrigues



Mestrado Integrado em Engenharia Informática
Universidade do Minho

Conteúdo

1	Introdução	2
2	Questões e Respostas	3
2.1	Parte I	3
2.1.1	Captura de Tráfego IP	3
2.2	Parte II	11
2.2.1	Endereçamento e Encaminhamento IP	11
2.2.2	Definição de sub-redes	23
3	Conclusão	29

1 Introdução

No âmbito da Unidade Curricular Redes de Computadores, foi nos proposto um trabalho dividido em 2 fases. Estas foram feitas durante as aulas Prático Laboratoriais.

Na primeira parte, foi realizado um estudo do registo de datagramas IP recebidos e enviados no decorrer do traceroute. Este estudo consiste na análise dos vários campos de um datagrama IP e no processo de fragmentação realizado pelo IP.

Na segunda fase continua-se o estudo do protocolo IPv4 com ênfase no endereçamento e encaminhamento IP.

2 Questões e Respostas

2.1 Parte I

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

2.1.1 Captura de Tráfego IP

Com o objetivo de obter um registo de tráfego IP, pretende-se usar o programa *traceroute* para descobrir uma rota IP, enviando pacotes de diferentes tamanhos para um determinado destino X.

Questões

1. Prepare uma topologia no *CORE* para verificar o comportamento do *traceroute*. Ligue um host (servidor) *s1* a um router *r2*; o router *r2* a um router *r3*, o router *r3* a um router *r4*, que por sua vez, se liga a um host (pc) *h5*.

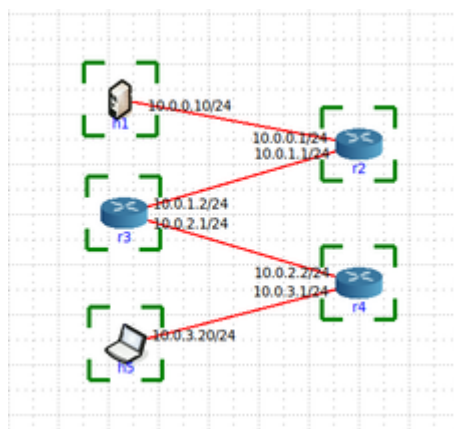


Figura 1: Topologia no *CORE*

a) Ative o wireshark ou o tcpdump no pc s1. Numa shell de s1, execute o comando `traceroute -I` para o endereço IP do host h5.

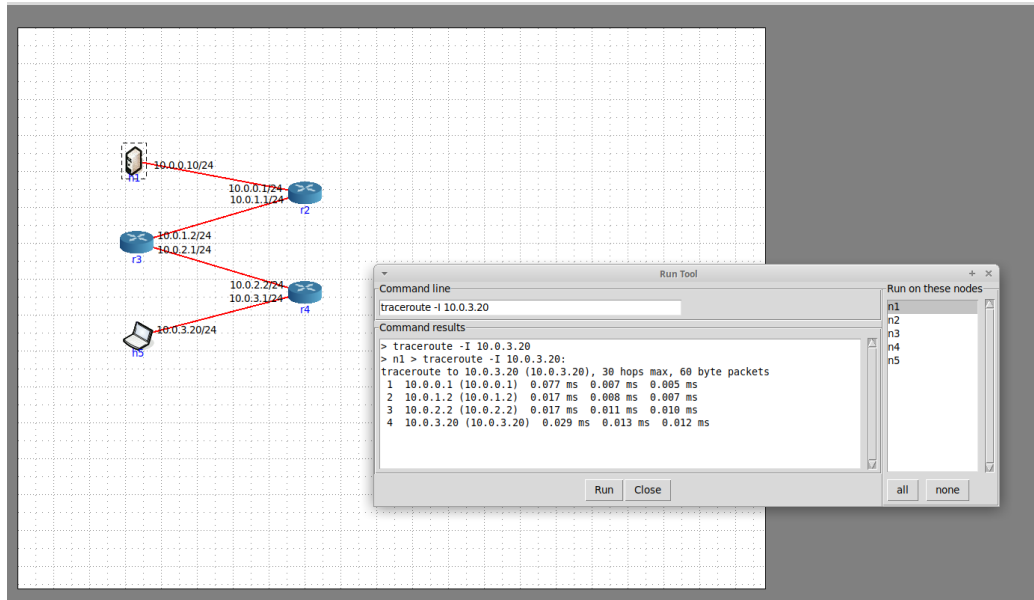


Figura 2: Execução do comando *traceroute*

b) Registre e analise o tráfego ICMP enviado por s1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

Resposta: Como esperado, todos os datagramas enviados com $TTL < 4$ não chegaram ao destino e receberam um *Time-to-live exceeded*. Todos os datagramas enviados com o $TTL \geq 4$ chegam ao destino e recebem uma resposta.

icmp					
No.	Time	Source	Destination	Protocol	Length/Info
9	21.695314014	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=1/256, ttl=1 (no response found!)
10	21.695326459	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
11	21.695335051	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=2/512, ttl=1 (no response found!)
12	21.695339855	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
13	21.695343604	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=3/768, ttl=1 (no response found!)
14	21.695347274	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
15	21.695351226	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=4/1024, ttl=2 (no response found!)
16	21.695365557	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
17	21.695370208	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=5/1280, ttl=2 (no response found!)
18	21.695376800	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
19	21.695386133	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=6/1536, ttl=2 (no response found!)
20	21.695386398	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
21	21.695396142	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=7/1792, ttl=3 (no response found!)
22	21.695407381	10.0.2.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
23	21.695410884	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=8/2048, ttl=3 (no response found!)
24	21.695419984	10.0.2.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
25	21.695423301	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=9/2304, ttl=3 (no response found!)
26	21.695432429	10.0.2.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
27	21.695436400	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=10/2560, ttl=4 (reply in 28)
28	21.695463576	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=10/2560, ttl=61 (request in 27)
29	21.695468066	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=11/2816, ttl=4 (reply in 30)
30	21.695479591	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=11/2816, ttl=61 (request in 29)
31	21.695482973	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=12/3072, ttl=4 (reply in 32)
32	21.695494839	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=12/3072, ttl=61 (request in 31)
33	21.695497915	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=13/3328, ttl=5 (reply in 34)
34	21.695508855	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=13/3328, ttl=61 (request in 33)
35	21.695512215	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=14/3584, ttl=5 (reply in 36)
36	21.695522891	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=14/3584, ttl=61 (request in 35)
37	21.695526123	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=15/3840, ttl=5 (reply in 38)
38	21.695537081	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=15/3840, ttl=61 (request in 37)
39	21.695546991	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=16/4096, ttl=6 (reply in 40)
40	21.695552620	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=16/4096, ttl=61 (request in 39)

Figura 3: Tráfego ICMP resultante da execução do comando *tracert*

c) Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino h5? Verifique na prática se a sua resposta está correta.

Resposta: O valor inicial mínimo do campo TTL deve ser 4. Analisando a Figura 4 verificamos que para todas as tentativas com o campo TTL menor do que 4 recebemos um *Time-to-live exceeded*.

icmp					
No.	Time	Source	Destination	Protocol	Length/Info
9	21.695314014	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=1/256, ttl=1 (no response found!)
10	21.695326459	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
11	21.695335051	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=2/512, ttl=1 (no response found!)
12	21.695339855	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
13	21.695343604	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=3/768, ttl=1 (no response found!)
14	21.695347274	10.0.0.1	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
15	21.695351226	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=4/1024, ttl=2 (no response found!)
16	21.695365557	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
17	21.695370208	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=5/1280, ttl=2 (no response found!)
18	21.695376800	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
19	21.695386133	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=6/1536, ttl=2 (no response found!)
20	21.695386398	10.0.1.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
21	21.695396142	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=7/1792, ttl=3 (no response found!)
22	21.695407381	10.0.2.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
23	21.695410884	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=8/2048, ttl=3 (no response found!)
24	21.695419984	10.0.2.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
25	21.695423301	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=9/2304, ttl=3 (no response found!)
26	21.695432429	10.0.2.2	10.0.0.10	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
27	21.695436400	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request id=0x001e, seq=10/2560, ttl=4 (reply in 28)
28	21.695463576	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply id=0x001e, seq=10/2560, ttl=61 (request in 27)

Figura 4: Verificação de valor mínimo do campo TTL

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

TTL	Pacote 1	Pacote 2	Pacote 3	Média
4	0.027 ms	0.012 ms	0.011 ms	0.016(6) ms
5	0.11 ms	0.11 ms	0.11 ms	0.011 ms
6	0.12 ms	-	-	0.012 ms

Média Final: 0.0135714286 ms

2. Pretende-se agora usar o traceroute na sua máquina nativa e gerar data-gramas de diferentes tamanhos. Usando o wireshark capture o tráfego gerado pelo traceroute, usando o tamanho por defeito. Utilize como máquina destino o host macro.uminho.pt. Selecione a primeira mensagem ICMP capturada.

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

Resposta: O endereço IP da interface ativa da minha máquina nativa é **10.0.2.15**.

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

Resposta: O valor do campo protocolo é **1** e identifica o protocolo ICMP.

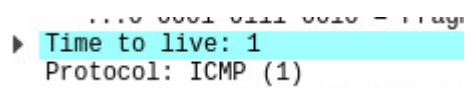


Figura 5: Valor do campo protocolo e identificação

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

Resposta: O cabeçalho IP(v4) contém **20 bytes** e o campo de dados (payload) do datagrama tem **40 bytes**. O cálculo do payload é feito da seguinte forma.

$$p = t - h$$

Onde p =tamanho do payload; t =tamanho total; h =tamanho do cabeçalho.

```

Internet Protocol Version 4, Src: 10.0.2.15, Dst: 193.150.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0x4958 (18776)
  ▶ Flags: 0x0000

```

Figura 6: Tamanho total e do cabeçalho IP(v4)

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

Resposta: O datagrama IP não foi fragmentado porque o campo *fragment offset* está a 0 e o *more fragments* está em *No Set*, logo significa que é o primeiro fragmento e que não há mais nenhum fragmento, logo este é o primeiro e o último fragmento, assim, concluímos que o datagrama não foi fragmentado.

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

Resposta: Após analisar a sequência ICMP concluímos que, no campo do cabeçalho IP, a **identificação** e o **TTL** variavam de pacote para pacote.

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

Resposta: São sempre **incrementados 1 valor**.

g) Ordene o tráfego capturado por endereços 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 aos seu host? Porquê?

Resposta: O valor do campo *TTL* não permanece constante. Foram encontrados 3 mensagens de resposta *ICMP TTL exceeded* com o campo **TTL = 255**, 3 mensagens com o **TTL = 64** e outras 3 com o **TTL = 254**.

3. Pretende-se agora analisar a fragmentação de pacotes IP. Observe depois do tamanho de pacote ter sido definido para 42XX bytes

Como somos o grupo 05, iremos utilizar o tamanho de pacote de 4205 bytes.

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

Resposta: Houve necessidade de fragmentar porque o pacote inicial continha 4205 bytes e o tamanho máximo que podemos enviar é 1500 bytes, então tivemos de fragmentar em 3.

```
▼ [3 IPv4 Fragments (4185 bytes): #1(1480), #2(1480), #4(1225)]
  [Frame: 1, payload: 0-1479 (1480 bytes)]
  [Frame: 2, payload: 1480-2959 (1480 bytes)]
  [Frame: 4, payload: 2960-4184 (1225 bytes)]
  [Fragment count: 3]
  [Reassembled IPv4 length: 4185]
  [Reassembled IPv4 data: 0800221c1459000148494a4b4c4d4e4f5051525354555657...]
```

Figura 7: Fragmentação do datagrama

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?

Resposta: A informação que indica que o datagrama foi fragmentado está no campo *more fragments*. Como este campo está em *set*, significa que o datagrama foi fragmentado. O facto do *fragment offset* estar a 0 significa que

este é o **primeiro** segmento. O tamanho do datagrama IP é **1480 bytes**, juntando com os 20 bytes do header, chegamos aos 1500 bytes do tamanho total.

```
▼ Internet Protocol Version 4, Src: 10.0.2.15, 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: 0x0943 (2371)
  ▼ 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: 0xb357 [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 193.136.9.240
    Reassembled IPv4 in frame: 4
  ► Data (1480 bytes)
```

Figura 8: Descrição do primeiro fragmento

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?*

Resposta: Não se trata do primeiro fragmento porque o *fragment offset* é maior que 0. Neste caso o *fragment offset* é 185, se fizermos os cálculos, verificamos que o *offset* corresponde ao tamanho do datagrama IP do primeiro fragmento, logo pertence ao **segundo fragmento**.

$$185 * 8 = 1480$$

Analisando o campo *more fragments* verificamos que está em *set* logo existem mais fragmentos.

```
▼ Internet Protocol Version 4, Src: 10.0.2.15, 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: 0x0943 (2371)
  ▼ 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: 0xb29e [validation disabled]
    [Header checksum status: Unverified]
    Source: 10.0.2.15
    Destination: 193.136.9.240
    Reassembled IPv4 in frame: 4
  ► Data (1480 bytes)
```

Figura 9: Descrição do segundo fragmento

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

Resposta: Foram criados **3 fragmentos**. O último fragmento é facilmente detetado a partir da análise do campo *more fragments*. Se este campo estiver em *No set*, significa que é o último segmento.

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*

Resposta: Os campos que mudam no cabeçalho IP são o checksum e o *fragment offset*. Assim, sabendo o *fragment offset*, sabemos a ordem de cada fragmento, sendo assim possível reconstruir o datagrama.

2.2 Parte II

Nesta secção continua-se o estudo do protocolo IPv4 com ênfase no endereçamento e encaminhamento IP. Serão estudadas algumas técnicas mais relevantes que foram propostas para aumentar a escalabilidade do protocolo IP, mitigar a exaustão dos endereços IPv4 e também reduzir os recursos de memória necessários nos *routers* para manter as tabelas de encaminhamento.

2.2.1 Endereçamento e Encaminhamento IP

Caso de estudo

Considere que a organização MIEI-RC é constituída por quatro departamentos (A, B, C e D) e cada departamento possui um *router* de acesso à sua rede local. Estes *routers* de acesso (R_a , R_b , R_c e R_d) estão interligados entre si por ligações Ethernet a 1 Gbps, formando um anel. Por sua vez, existe um servidor (S_1) na rede do departamento A e, pelo menos, três *laptops* por departamento, interligados ao *router* respetivo através de um comutador (*switch*). S_1 tem uma ligação a 1 Gbps e os *laptops* ligações de 100 Mbps. Considere apenas a existência de um comutador por departamento.

A conectividade IP externa da organização é assegurada através de um router de acesso R_e conectado a um R_d por uma ligação ponto-a-ponto a 10 Gbps.

Questões

1. *Atenda aos endereços IP e máscaras de rede que foram atribuídos pelo CORE aos diversos equipamentos da topologia.*

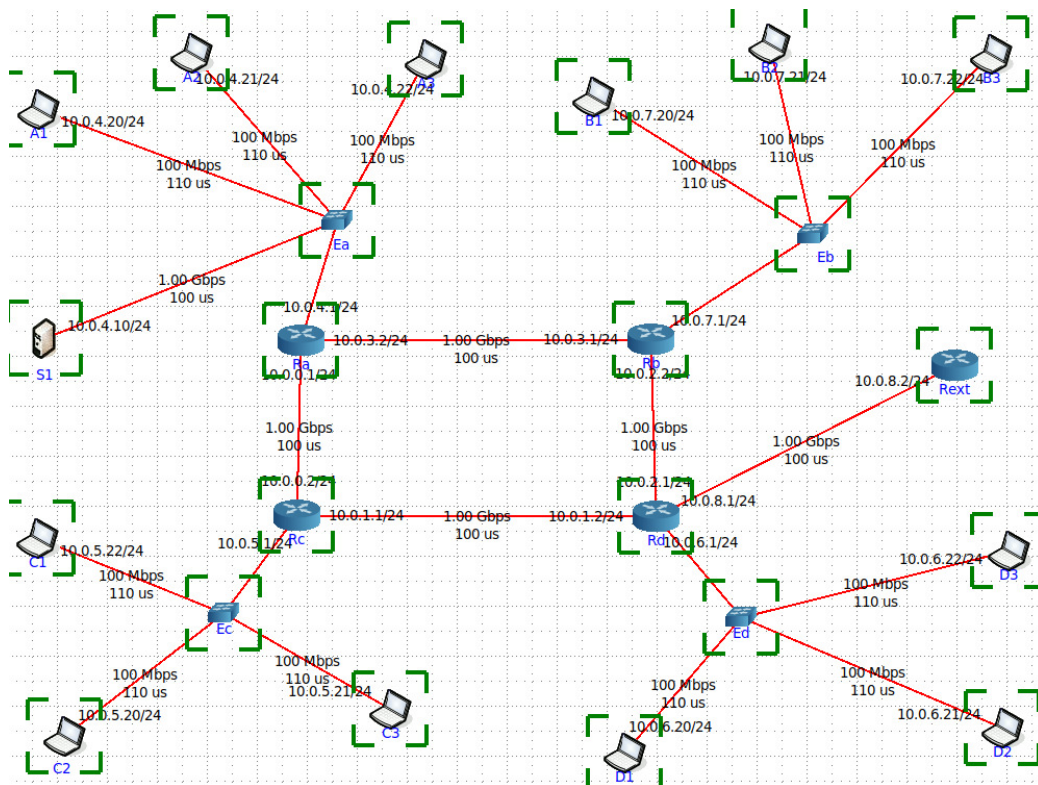


Figura 10: Topologia CORE

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.

Resposta: A Figura 11 mostra os endereços IP e as máscaras de rede atribuídas pelo CORE, neste caso, a máscara de rede usada é igual para todos os equipamentos: $\backslash 24$ ou seja 255.255.255.0

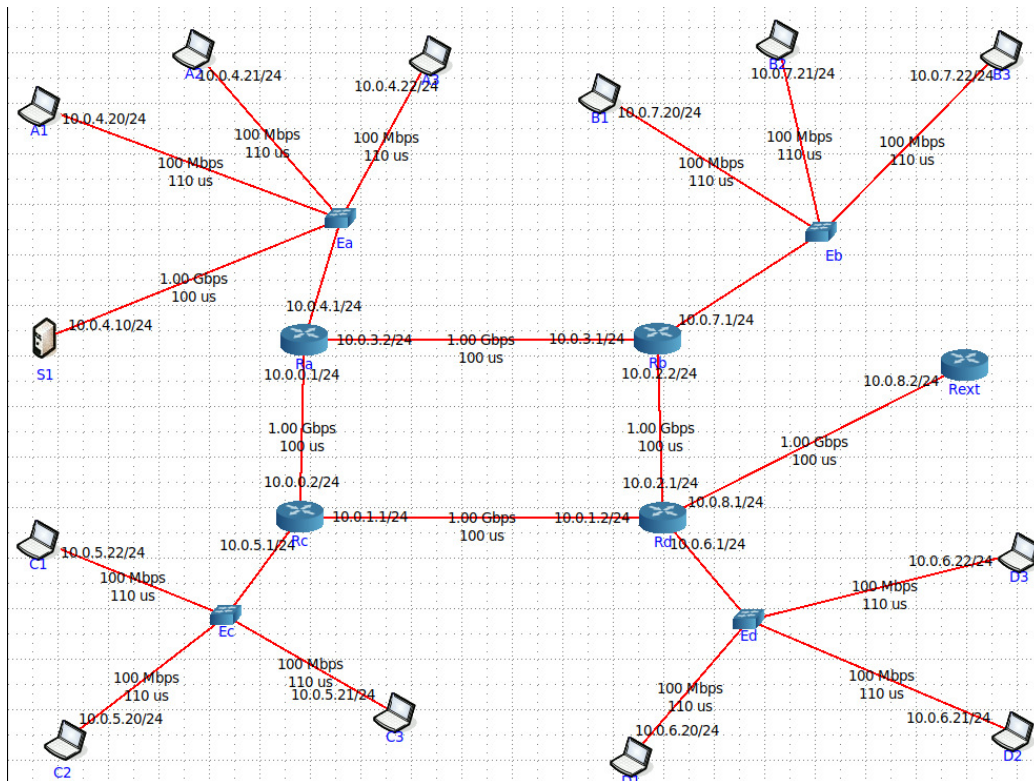


Figura 11: Endereços atribuídos pelo *CORE*

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

Resposta: Tratam-se de endereços *privados* pois pertencem ao bloco de endereços entre $10.0.0.0 - 10.255.255.255 /8$.

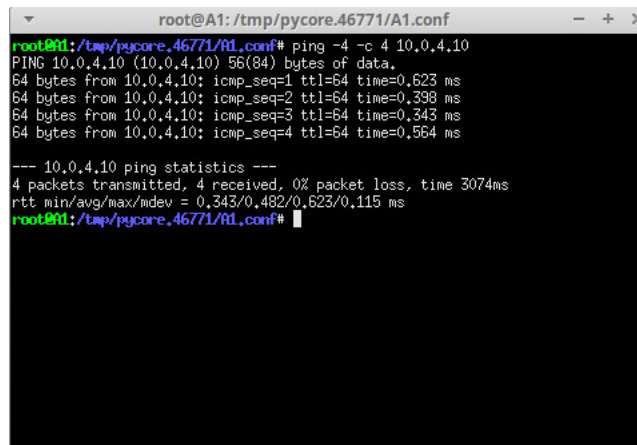
c) *Por que razão não é atribuído um endereço IP aos switches?*

Resposta: Não é atribuído um endereço de IP aos *switches* porque estes são usados em ligações de nível 2 e o endereço de IP apenas é atribuído em ligações de nível 3.

d) *Usando o comando ping certifique-se que existe conectividade IP entre os laptops dos vários departamentos e o servidor do departamento A (basta*

certificar-se da conectividade de um laptop por departamento.)

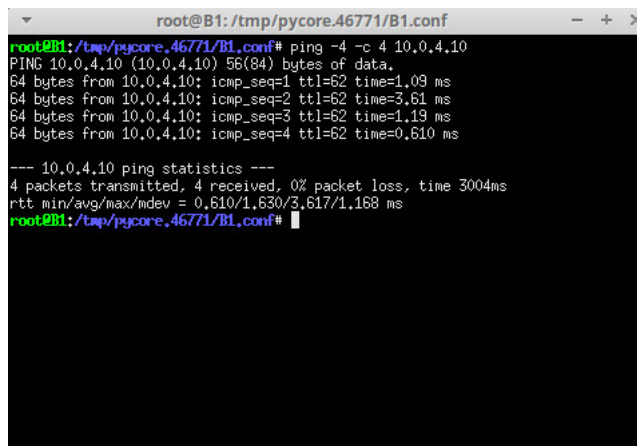
Resposta: Para provar que existe conectividade IP entre os *laptops* e o servidor do departamento A, basta executar o comando `ping -4 -c 4 10.0.4.10` num *laptop* de cada departamento. Onde `10.0.4.10` corresponde ao endereço IP do servidor do departamento A.



```
root@A1:/tmp/pycore.46771/A1.conf
root@A1:/tmp/pycore.46771/A1.conf# ping -4 -c 4 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=0.623 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=64 time=0.398 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=64 time=0.343 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=64 time=0.564 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.343/0.482/0.623/0.115 ms
root@A1:/tmp/pycore.46771/A1.conf#
```

Figura 12: Resultado da execução do comando `ping -4 -c 4 10.0.4.10` na shell do **A1**



```
root@B1:/tmp/pycore.46771/B1.conf
root@B1:/tmp/pycore.46771/B1.conf# ping -4 -c 4 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=1.09 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=62 time=3.61 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=62 time=1.19 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=62 time=0.610 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.610/1.630/3.617/1.168 ms
root@B1:/tmp/pycore.46771/B1.conf#
```

Figura 13: Resultado da execução do comando `ping -4 -c 4 10.0.4.10` na shell do **B1**

```
root@C2: /tmp/pycore.46771/C2.conf
root@C2: /tmp/pycore.46771/C2.conf# ping -4 -c 4 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=1.17 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=62 time=0.519 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=62 time=0.816 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=62 time=0.646 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3013ms
rtt min/avg/max/mdev = 0.519/0.789/1.178/0.250 ms
root@C2: /tmp/pycore.46771/C2.conf#
```

Figura 14: Resultado da execução do comando `ping -4 -c 4 10.0.4.10` na shell do **C2**

```
root@D3: /tmp/pycore.46771/D3.conf
root@D3: /tmp/pycore.46771/D3.conf# ping -4 -c 4 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data:
64 bytes from 10.0.4.10: icmp_seq=1 ttl=61 time=1.25 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=61 time=0.986 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=61 time=0.972 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=61 time=0.816 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 0.816/1.007/1.257/0.163 ms
root@D3: /tmp/pycore.46771/D3.conf#
```

Figura 15: Resultado da execução do comando `ping -4 -c 4 10.0.4.10` na shell do **D3**

e) Verifique se existe conectividade IP do router de acesso R_e para o servidor S_1 .

Resposta: Sim, existe conectividade IP do **Rext** para o servidor **S1**.


```
root@Rext:/tmp/pycore.46771/Rext.conf
root@Rext:/tmp/pycore.46771/Rext.conf# ping -4 -c 4 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data:
64 bytes from 10.0.4.10: icmp_seq=1 ttl=61 time=1.84 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=61 time=0.921 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=61 time=1.20 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=61 time=0.857 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3027ms
rtt min/avg/max/mdev = 0.857/1.205/1.841/0.391 ms
root@Rext:/tmp/pycore.46771/Rext.conf#
```

Figura 16: Resultado da execução do comando `ping -4 -c 4 10.0.4.10` na shell do router para o exterior

2. Para o router e um laptop do departamento B:

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

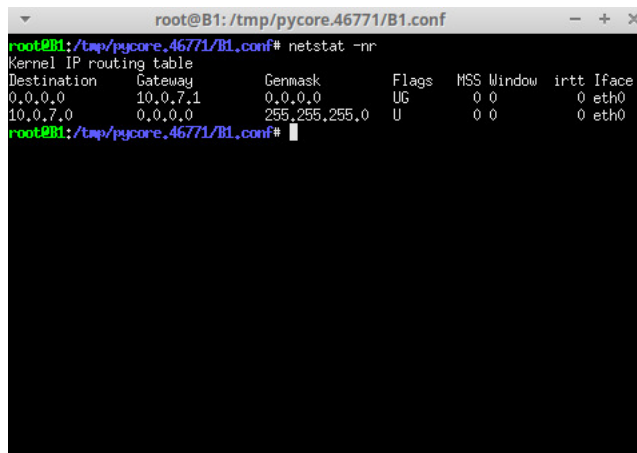
```
root@Rb:/tmp/pycore.46771/Rb.conf
root@Rb:/tmp/pycore.46771/Rb.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 10.0.3.2 255.255.255.0 UG 0 0 0 eth1
10.0.1.0 10.0.2.1 255.255.255.0 UG 0 0 0 eth0
10.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.3.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
10.0.4.0 10.0.3.2 255.255.255.0 UG 0 0 0 eth1
10.0.5.0 10.0.2.1 255.255.255.0 UG 0 0 0 eth0
10.0.6.0 10.0.2.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
10.0.8.0 10.0.2.1 255.255.255.0 UG 0 0 0 eth0
root@Rb:/tmp/pycore.46771/Rb.conf#
```

Figura 17: Resultado da execução do comando `netstat -rn` na shell do **Rb**

Resposta: Cada linha da tabela lê-se da mesma forma, assim, a leitura da primeira linha seria a seguinte:

Um datagrama destinado à rede 10.0.0.0 será entregue na interface de endereço 10.0.3.2 saindo pela interface local eth1.

O processo é igual para o resto das linhas.



```
root@B1: /tmp/pycore.46771/B1.conf
root@B1: /tmp/pycore.46771/B1.conf# netstat -nr
Kernel IP routing table
Destination    Gateway         Genmask         Flags   MSS Window  irtt  Iface
0.0.0.0         10.0.7.1        0.0.0.0         UG        0  0        0   eth0
10.0.7.0        0.0.0.0         255.255.255.0   U        0  0        0   eth0
root@B1: /tmp/pycore.46771/B1.conf#
```

Figura 18: Resultado da execução do comando *netstat -rn* na shell do **B1**

b) *Diga justificando, se está a ser usado encaminhamento estático ou dinâmico.*

Resposta: Observando as figuras 19 e 20, reparamos que os *routers* contêm processos como **ospf** que indicam que o encaminhamento é **dinâmico**. Pelo contrário, nos equipamentos não temos nenhum processo destes a correr, logo o encaminhamento nestes casos é **estático**.

```
root@A1: /tmp/pycore.39763/A1.conf
root@A1: /tmp/pycore.39763/A1.conf# ps -e
PID TTY          TIME CMD
  1 ?            00:00:00 vncd
 19 pts/2        00:00:00 bash
 30 pts/2        00:00:00 ps
root@A1: /tmp/pycore.39763/A1.conf#
```

Figura 19: Processos de qualquer laptop do sistema.

```
root@Ra: /tmp/pycore.39763/Ra.conf
root@Ra: /tmp/pycore.39763/Ra.conf# ps -e
PID TTY          TIME CMD
  1 ?            00:00:00 vncd
 58 ?            00:00:00 zebra
 64 ?            00:00:00 ospfd
 68 ?            00:00:00 ospfd
 76 pts/2        00:00:00 bash
 84 pts/2        00:00:00 ps
root@Ra: /tmp/pycore.39763/Ra.conf#
```

Figura 20: Processos de qualquer router do sistema.

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 S_1 localizado no departamento A. Use o comando `route delete` para o efeito. Que implicação tem esta medida para os utilizadores da empresa que acedem ao servidor?

```
root@S1: /tmp/pycore.45777/S1.conf
root@S1: /tmp/pycore.45777/S1.conf# route delete default
root@S1: /tmp/pycore.45777/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@S1: /tmp/pycore.45777/S1.conf#
```

Figura 21: Execução do comando para retirar a rota *default* da tabela de encaminhamento do servidor S1

Resposta: O facto da rota *default* ter sido eliminada da tabela de encaminhamento do servidor S_1 faz com que seja impossível aceder o servidor a partir de outro departamento que não seja o departamento A, onde este servidor está presente.

```
root@C2: /tmp/pycore.40555/C2.conf
root@C2: /tmp/pycore.40555/C2.conf# ping -4 -c 4 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
^C
--- 10.0.4.10 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3069ms
root@C2: /tmp/pycore.40555/C2.conf#
```

Figura 22: Tentativa de conexão de um *laptop* do departamento C com o servidor S1

d) *Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor S_1 por forma a contornar a restrição imposta na alínea c).*

Utilize para o efeito o comando `route add` e registre os comandos que usou.

```
root@S1: /tmp/pycore.40555/S1.conf
root@S1: /tmp/pycore.40555/S1.conf# route add -net 10.0.5.0 gw 10.0.4.1 netmask 255.255.255.0
root@S1: /tmp/pycore.40555/S1.conf# route add -net 10.0.6.0 gw 10.0.4.1 netmask 255.255.255.0
root@S1: /tmp/pycore.40555/S1.conf# route add -net 10.0.7.0 gw 10.0.4.1 netmask 255.255.255.0
root@S1: /tmp/pycore.40555/S1.conf# route add -net 10.0.8.0 gw 10.0.4.1 netmask 255.255.255.0
root@S1: /tmp/pycore.40555/S1.conf#
```

Figura 23: Sequência de comandos utilizados para restaurar a conectividade para o servidor S1

e) *Teste a nova política de encaminhamento garantindo que o servidor está novamente acessível utilizando para o efeito o comando `ping`. Registe a nova tabela de encaminhamento do servidor.*

```
root@S1: /tmp/pycore.40555/S1.conf
root@S1: /tmp/pycore.40555/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.5.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.6.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
10.0.8.0 10.0.4.1 255.255.255.0 UG 0 0 0 eth0
root@S1: /tmp/pycore.40555/S1.conf#
```

Figura 24: Tabela de encaminhamento depois de adicionar as rotas da alínea anterior

```
root@A1: /tmp/pycore.40555/A1.conf
root@A1: /tmp/pycore.40555/A1.conf# ping -c 4 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=3.78 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=64 time=1.55 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=64 time=1.43 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=64 time=1.92 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.435/2.174/3.789/0.950 ms
root@A1: /tmp/pycore.40555/A1.conf#
```

Figura 25: Conexão entre *laptop* do departamento A com o servidor S1.

```
root@B3: /tmp/pycore.40555/B3.conf
root@B3: /tmp/pycore.40555/B3.conf# ping -c 4 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=4.30 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=62 time=2.42 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=62 time=1.98 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=62 time=3.22 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 1.986/2.984/4.305/0.884 ms
root@B3: /tmp/pycore.40555/B3.conf#
```

Figura 26: Conexão entre *laptop* do departamento B com o servidor S1.

```
root@C2: /tmp/pycore.40555/C2.conf
root@C2: /tmp/pycore.40555/C2.conf# ping -c 4 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=5.79 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=62 time=2.55 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=62 time=3.88 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=62 time=0.691 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 0.691/3.229/5.790/1.863 ms
root@C2: /tmp/pycore.40555/C2.conf#
```

Figura 27: Conexão entre *laptop* do departamento C com o servidor S1.

```
root@D1: /tmp/pycore.40555/D1.conf
root@D1: /tmp/pycore.40555/D1.conf# ping -c 4 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_seq=1 ttl=61 time=6.17 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=61 time=2.97 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=61 time=5.81 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=61 time=4.67 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 2.979/4.908/6.170/1.245 ms
root@D1: /tmp/pycore.40555/D1.conf#
```

Figura 28: Conexão entre *laptop* do departamento D com o servidor S1.

```
root@Rext: /tmp/pycore.40555/Rext.conf
root@Rext: /tmp/pycore.40555/Rext.conf# ping -4 -c 4 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data:
64 bytes from 10.0.4.10: icmp_seq=1 ttl=61 time=6.82 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=61 time=6.16 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=61 time=6.81 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=61 time=6.80 ms

--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 6.166/6.651/6.820/0.286 ms
root@Rext: /tmp/pycore.40555/Rext.conf# $
```

Figura 29: Conexão entre o router do exterior com o servidor S1.

2.2.2 Definição de sub-redes

3. Por forma a minimizar a falta de endereços IPv4 é comum a utilização de sub-redes. Além disso, a definição de sub-redes permite uma melhor organização do espaço de endereçamentos das redes em questão.

Considere a topologia definida anteriormente. Assuma que o endereçamento entre os router se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.

1) Considere que dispões apenas do endereço de rede IP 172.yyx.32.0/20, em que "yy" são os dígitos correspondendo ao seu número de grupo (Gyy) e "x" é o dígito correspondente ao seu turno prático (PLx). 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.

Como somos G05 e PL2, então iremos usar o endereço de rede IP 172.52.32.0/20.

Resposta: Vamos ter 4 subredes, uma por departamento.

Se dividirmos o endereço IP pelas suas componentes, reparamos que **20 bits** são usados para identificar a **rede**, agora precisamos de calcular o número de bits que iremos usar para identificar a **sub-rede**.

$$2^3 - 2 = 6$$

Como 3 bits são suficientes para representar 6 sub-redes diferentes, então iremos usar 3 bits para identificar a sub-rede. Assim a composição do endereço IP de cada sub-rede será **20 bits para a identificar a rede, 3 bits para identificar a sub-rede, 9 bits para identificar o *host***.

$$2^9 - 2 = 510$$

Com esta composição podemos identificar 510 *hosts* diferentes, mais do que suficientes para este projeto.

Então, para cada sub-rede, teremos de alterar o endereço IP de cada equipamento, assim teremos:

SR1: Departamento A

S1 - 172.52.32.10/23

A1 - 172.52.32.20/23

A2 - 172.52.32.21/23

A3 - 172.52.32.22/23

Ra - 172.52.32.1/23

SR2: Departamento B

B1 - 172.52.34.20/23

B2 - 172.52.34.21/23

B3 - 172.52.34.22/23

Rb - 172.52.34.1/23

SR3: Departamento C

C1 - 172.52.36.22/23

C2 - 172.52.36.20/23

Rc - 172.52.36.1/23

SR4: Departamento D

D1 - 172.52.38.20/23

D2 - 172.52.38.21/23

D3 - 172.52.38.22/23

Rd - 172.52.38.1/23

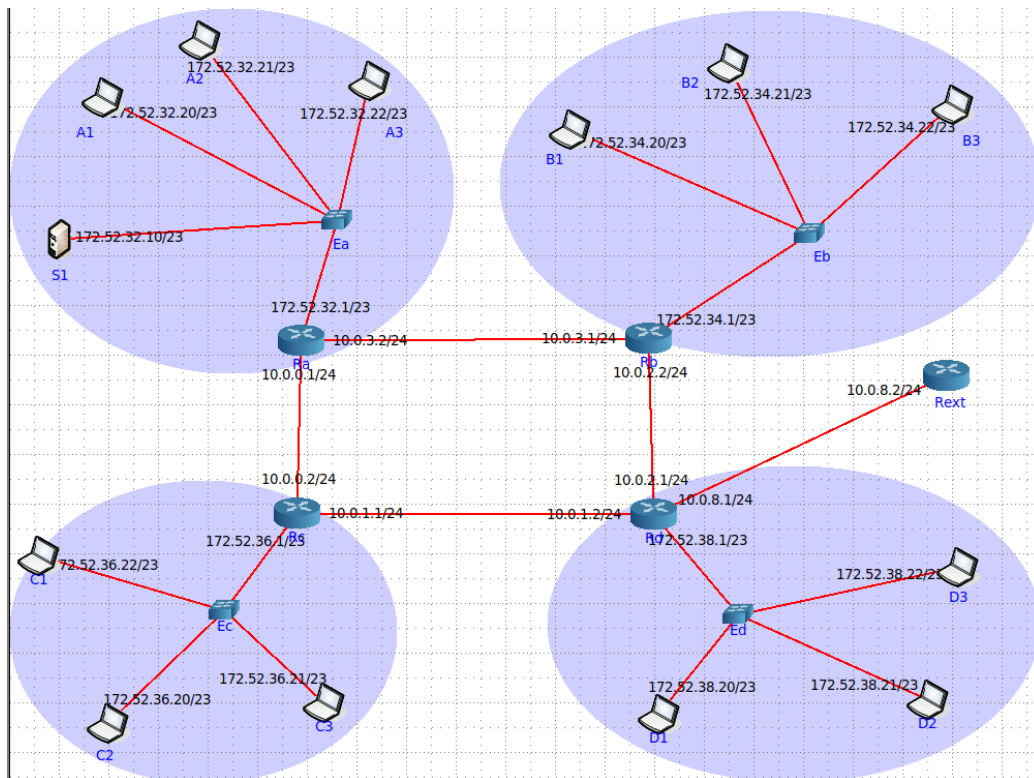


Figura 30: Novo endereçamento, contendo sub-redes.

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

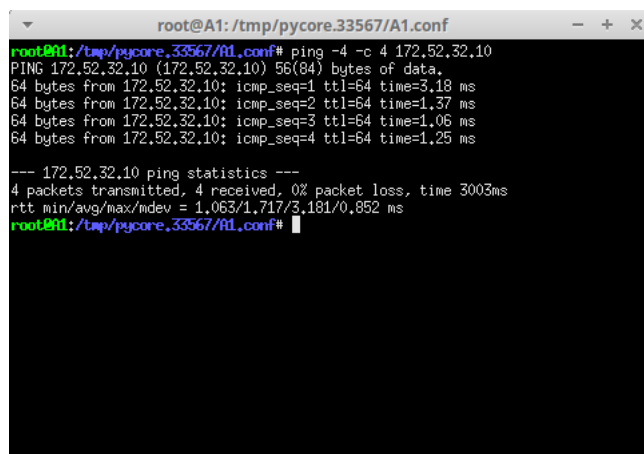
Resposta: A máscara de rede em decimal é **255.255.254.0** e as interfaces podem ser calculadas através da seguinte fórmula:

$$2^9 - 2 = 510$$

Então, podemos interligar **510 interfaces em cada departamento**.

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

Resposta: Para provar que existe a conectividade utilizamos o comando *ping* para o servidor de pelo menos um dos *laptops* de cada departamento.

A terminal window titled 'root@A1: /tmp/pycore.33567/A1.conf' with standard window controls. The terminal shows the execution of the command 'ping -c 4 172.52.32.10'. The output displays four successful ping responses with varying times (3.18 ms, 1.37 ms, 1.06 ms, 1.25 ms) and a summary statistics block indicating 0% packet loss and a total time of 3003ms.

```
root@A1: /tmp/pycore.33567/A1.conf# ping -c 4 172.52.32.10
PING 172.52.32.10 (172.52.32.10) 56(84) bytes of data:
64 bytes from 172.52.32.10: icmp_seq=1 ttl=64 time=3.18 ms
64 bytes from 172.52.32.10: icmp_seq=2 ttl=64 time=1.37 ms
64 bytes from 172.52.32.10: icmp_seq=3 ttl=64 time=1.06 ms
64 bytes from 172.52.32.10: icmp_seq=4 ttl=64 time=1.25 ms

--- 172.52.32.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 1.063/1.717/3.181/0.852 ms
root@A1: /tmp/pycore.33567/A1.conf#
```

Figura 31: Execução do comando *ping* para o servidor de um *laptop* do departamento A.

```
root@B1: /tmp/pycore.33567/B1.conf
root@B1: /tmp/pycore.33567/B1.conf# ping -c 4 172.52.32.10
PING 172.52.32.10 (172.52.32.10) 56(84) bytes of data:
64 bytes from 172.52.32.10: icmp_seq=1 ttl=62 time=4.61 ms
64 bytes from 172.52.32.10: icmp_seq=2 ttl=62 time=3.40 ms
64 bytes from 172.52.32.10: icmp_seq=3 ttl=62 time=3.36 ms
64 bytes from 172.52.32.10: icmp_seq=4 ttl=62 time=2.92 ms

--- 172.52.32.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 2.924/3.574/4.613/0.630 ms
root@B1: /tmp/pycore.33567/B1.conf#
```

Figura 32: Execução do comando *ping* para o servidor de um *laptop* do departamento B.

```
root@C1: /tmp/pycore.33567/C1.conf
root@C1: /tmp/pycore.33567/C1.conf# ping -c 4 172.52.32.10
PING 172.52.32.10 (172.52.32.10) 56(84) bytes of data:
64 bytes from 172.52.32.10: icmp_seq=1 ttl=62 time=4.35 ms
64 bytes from 172.52.32.10: icmp_seq=2 ttl=62 time=3.60 ms
64 bytes from 172.52.32.10: icmp_seq=3 ttl=62 time=3.40 ms
64 bytes from 172.52.32.10: icmp_seq=4 ttl=62 time=3.35 ms

--- 172.52.32.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 3.352/3.680/4.358/0.402 ms
root@C1: /tmp/pycore.33567/C1.conf#
```

Figura 33: Execução do comando *ping* para o servidor de um *laptop* do departamento C.

```
root@D1: /tmp/pycore.33567/D1.conf
root@D1: /tmp/pycore.33567/D1.conf# ping -c 4 172.52.32.10
PING 172.52.32.10 (172.52.32.10) 56(84) bytes of data:
64 bytes from 172.52.32.10: icmp_seq=1 ttl=61 time=6.22 ms
64 bytes from 172.52.32.10: icmp_seq=2 ttl=61 time=5.84 ms
64 bytes from 172.52.32.10: icmp_seq=3 ttl=61 time=5.30 ms
64 bytes from 172.52.32.10: icmp_seq=4 ttl=61 time=5.25 ms

--- 172.52.32.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 5.259/5.659/6.229/0.404 ms
root@D1: /tmp/pycore.33567/D1.conf# s
```

Figura 34: Execução do comando *ping* para o servidor de um *laptop* do departamento D.

```
root@Rext: /tmp/pycore.33567/Rext.conf
root@Rext: /tmp/pycore.33567/Rext.conf# ping -c 4 172.52.32.10
PING 172.52.32.10 (172.52.32.10) 56(84) bytes of data:
64 bytes from 172.52.32.10: icmp_seq=1 ttl=61 time=6.21 ms
64 bytes from 172.52.32.10: icmp_seq=2 ttl=61 time=6.81 ms
64 bytes from 172.52.32.10: icmp_seq=3 ttl=61 time=14.1 ms
64 bytes from 172.52.32.10: icmp_seq=4 ttl=61 time=6.28 ms

--- 172.52.32.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 6.213/8.369/14.161/3.352 ms
root@Rext: /tmp/pycore.33567/Rext.conf#
```

Figura 35: Execução do comando *ping* para o servidor do *router* do exterior.

3 Conclusão

Concluído o trabalho, entendemos que houve uma aprendizagem notória relativamente aos conhecimentos adquiridos nesta Unidade Curricular:

Captura de Tráfego IP:

- Analisar tráfego através do WireShark;
- Filtragem de protocolos ICMP;
- Análise de fragmentação do datagrama.

Endereçamento e Encaminhamento IP:

- Identificação da mascara e de endereçamentos IP;
- Leitura de tabelas de encaminhamento;
- Verificação da conectividade;
- Eliminação e adição de rotas.

Definição de sub-redes:

- Cálculo do número de bits para a identificação da subrede;
- Atribuição de novos endereços;
- Identificação da nova mascara;
- Número máximo de interfaces possíveis de cada subrede.