

Trabalho Prático 2

Alexandra Santos, Inês Ferreira e Joana Branco
Universidade do Minho, Departamento de Informática
email: {a94523, a97372, a96584}@alunos.uminho.pt

1ª parte

Exercício 1

Prepare uma topologia CORE para verificar o comportamento do traceroute. Na topologia deve existir: um host (pc) cliente designado Bela cujo router de acesso é R2; o router R2 está simultaneamente ligado a dois routers R3 e R4; estes estão conectados a um router R5, que por sua vez, se liga a um host (servidor) designado Monstro. Ajuste o nome dos equipamentos atribuídos por defeito para o enunciado. Nas ligações (links) da rede de core estabeleça um tempo de propagação de 10ms. Após ativar a topologia, note que pode não existir conectividade IP imediata entre a Bela e o Monstro até que o anúncio de rotas entre routers estabilize.

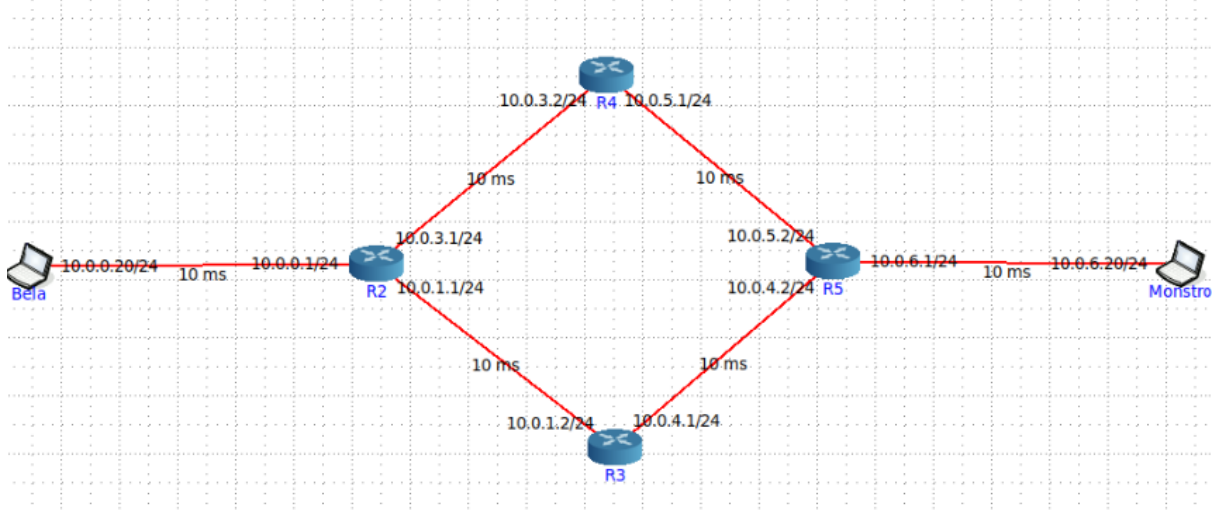


Figura 1: Core de acordo com a topologia

- A. Ative 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.38225/Bela.conf# traceroute -I 10.0.6.20
traceroute to 10.0.6.20 (10.0.6.20), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 21.358 ms 21.307 ms 21.301 ms
 2 10.0.1.2 (10.0.1.2) 42.144 ms 42.141 ms 42.138 ms
 3 10.0.4.2 (10.0.4.2) 63.579 ms 63.575 ms 63.572 ms
 4 10.0.6.20 (10.0.6.20) 85.110 ms 85.108 ms 85.104 ms
root@Bela:/tmp/pycore.38225/Bela.conf#
```

Figura 2: Terminal com o comando traceroute -I 10.0.6.20

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.

R: Através do tráfego ICMP, o host Bela envia três pacotes ao servidor Monstro. O TTL é apresentado de forma crescente e inicia-se com o valor de 1.

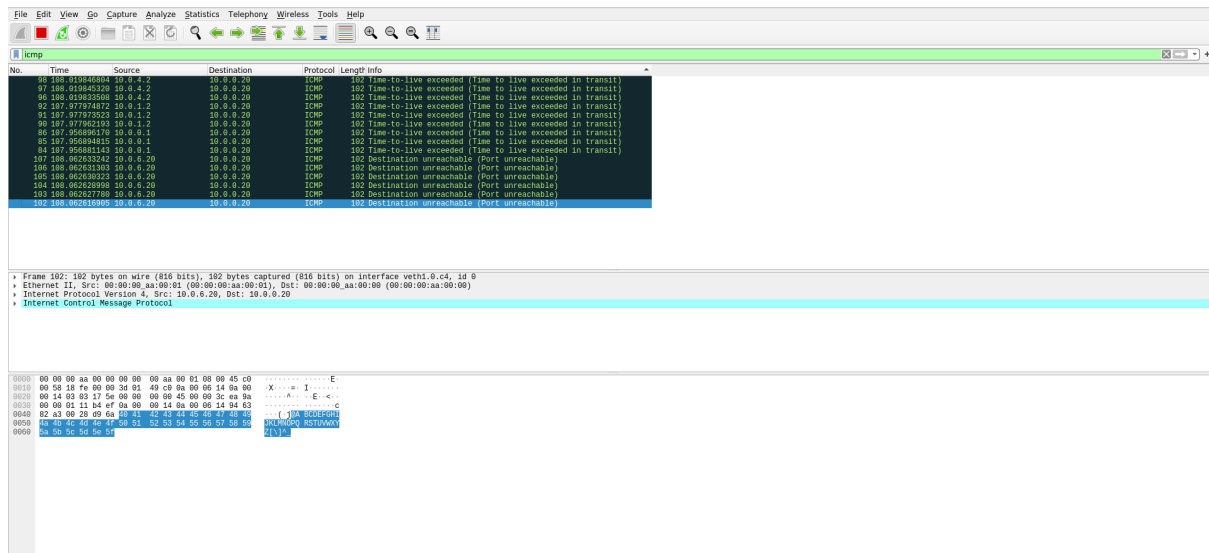


Figura 3: Tráfego ICMP no WireShark

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.

R: De maneira a alcançar o servidor Monstro, é necessário um valor mínimo de TTL correspondente a 4. Só é a partir deste mesmo valor que o servidor Monstro recebe uma resposta.

3382	604975.30123..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=5/1280, ttl=2 (no respons...
3383	604975.30124..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=6/1536, ttl=2 (no respons...
3384	604975.30124..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=7/1792, ttl=3 (no respons...
3385	604975.30125..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=8/2048, ttl=3 (no respons...
3386	604975.30125..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=9/2304, ttl=3 (no respons...
3387	604975.30126..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=10/2560, ttl=4 (reply in ...
3388	604975.30126..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=11/2816, ttl=4 (reply in ...
3389	604975.30127..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=12/3072, ttl=4 (reply in ...
3390	604975.30127..	10.0.0.20	10.0.6.20	ICMP	74 Echo (ping) request	id=0x002e, seq=13/3328, ttl=5 (reply in ...

Figura 4: Menor valor de TTL

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.

R: Ao aceder ao servidor, o valor RTT obtido foi de 93.216 ms. Comparando com os valores máximo e mínimo resultantes de 10 pacotes, podemos concluir que o resultado tende para o de menor valor.

```

root@Bela:/tmp/pycore.41931/Bela.conf# traceroute -I 10.0.6.20 -q 10
traceroute to 10.0.6.20 (10.0.6.20), 30 hops max, 60 byte packets
 1 10.0.0.1 (10.0.0.1) 40,750 ms 40,720 ms 40,718 ms 40,718 ms 40,717 ms
40,716 ms * * * *
 2 10.0.1.2 (10.0.1.2) 81,909 ms 81,910 ms 81,909 ms 81,907 ms 81,908 ms
81,907 ms * * * *
 3 10.0.4.2 (10.0.4.2) 83,062 ms 83,055 ms 61,635 ms 61,580 ms 61,570 ms
61,562 ms * * * *
 4 10.0.6.20 (10.0.6.20) 104,082 ms 104,070 ms 99,737 ms 99,686 ms 87,891
ms 87,820 ms 87,807 ms 87,798 ms 86,653 ms 86,619 ms
root@Bela:/tmp/pycore.41931/Bela.conf#

```

Figura 5: Terminal com o comando -q

- 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?

R: Não, o valor médio de atraso pode ser diferente em cada um dos sentidos. Ou seja, o tempo de ida do pacote pode ser maior que o tempo de volta, ou vice-versa logo, não pode ser calculado com precisão dividindo o RTT por dois.

Exercício 2

Pretende-se agora usar o traceroute na sua máquina nativa, e gerar datagramas IP de diferentes tamanhos.

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

R: O endereço IP da interface ativa é 172.26.254.254.

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

R: O valor do campo protocolo corresponde ao ICMP e permite identificar a mensagem capturada.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17498/23188, ttl=1 (no response found!)
2	0.002747	172.26.254.254	172.26.34.32	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
3	0.058765	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17499/23184, ttl=2 (no response found!)
4	0.070512	172.26.34.32	172.26.34.32	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
5	0.181389	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17500/23620, ttl=3 (reply in 6)
6	0.184733	193.136.9.254	172.26.34.32	ICMP	70	Echo (ping) reply id=0xb001, seq=17500/23620, ttl=253 (request in 5)
7	0.344850	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17501/23876, ttl=255 (reply in 8)
8	0.348151	193.136.9.254	172.26.34.32	ICMP	70	Echo (ping) reply id=0xb001, seq=17501/23876, ttl=255 (request in 7)
9	0.395576	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17502/24132, ttl=1 (no response found!)
10	0.401436	172.26.254.254	172.26.34.32	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
11	0.445619	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17503/24388, ttl=2 (no response found!)
12	0.451181	172.26.34.32	172.26.34.32	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
13	0.490161	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17504/24644, ttl=3 (reply in 14)
14	0.490416	193.136.9.254	172.26.34.32	ICMP	70	Echo (ping) reply id=0xb001, seq=17504/24644, ttl=253 (request in 13)
15	1.526773	162.159.136.234	172.26.34.32	TLSv1.2	169	Application Data
16	1.566948	172.26.34.32	162.159.136.234	TCP	54	53393 -> 443 [ACK] Seq=1 Ack=116 Win=500 Len=0
17	1.568760	162.159.136.234	172.26.34.32	TLSv1.2	135	Application Data
18	1.608687	172.26.34.32	162.159.136.234	TCP	54	53393 -> 443 [ACK] Seq=1 Ack=197 Win=500 Len=0
19	1.616892	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17505/24900, ttl=255 (reply in 20)
20	1.620348	193.136.9.254	172.26.34.32	ICMP	70	Echo (ping) reply id=0xb001, seq=17505/24900, ttl=253 (request in 19)
21	1.667305	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0xb001, seq=17506/25156, ttl=1 (no response found!)
22	1.671057	172.26.254.254	172.26.34.32	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)

< Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{F81E2681-F508-4584-0790-412AD4AF8693}, id 0
 > Ethernet II, Src: Liteontc_51:00:09 (3c:95:09:51:00:09), Dst: ComEdInt_ff:94:00 (00:00:03:ff:94:00)
 > Internet Protocol Version 4, Src: 172.26.34.32, Dst: 193.136.9.254
 > Internet Control Message Protocol

Figura 6: Primeira mensagem ICMP capturada

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

R: O cabeçalho IPv4 tem 20 bytes e são 56 bytes para o campo de dados, de comprimento total. Logo, o tamanho do payload é de 36 bytes (56-20=36).

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0x0001, seq=17498/23108, ttl=1 (no response found)
2	0.002427	172.26.154.254	172.26.34.32	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
3	0.005076	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0x0001, seq=17499/23109, ttl=2 (no response found)
4	0.005613	172.16.2.1	172.26.34.32	ICMP	70	Time-to-live exceeded (time to live exceeded in transit)
5	0.101300	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0x0001, seq=17500/23110, ttl=3 (reply in 6)
6	0.104733	193.136.9.254	172.26.34.32	ICMP	70	Echo (ping) reply id=0x0001, seq=17500/23110, ttl=253 (request in 5)
7	0.344850	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0x0001, seq=17501/23111, ttl=255 (reply in 8)
8	0.348161	193.136.9.254	172.26.34.32	ICMP	70	Echo (ping) reply id=0x0001, seq=17501/23111, ttl=253 (request in 7)
9	0.360176	172.26.34.32	193.136.9.254	ICMP	70	Echo (ping) request id=0x0001, seq=17502/23112, ttl=1 (no response found)


```

> Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{F81E2681-F598-4584-9790-412AD4AF0693}, id 0
> Ethernet II, Src: LixionTe_S1166169 (3c:95:09:51:66:69), Dst: CondaInt_FF:94:00 (08:00:03:ff:94:00)
  Internet Protocol Version 4, Src: 172.26.34.32, Dst: 193.136.9.254
    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 ECH-Capable Transport (0)
    Total Length: 56
    Identification: 0x0040 (26044)
    Flags: 0x00
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..0. .... = More fragments: Not set
      ...0 0000 0000 = Fragment Offset: 0
    Time to Live: 1
      [Expert Info (Note/Sequence): "Time To Live" only 1]
    Protocol: ICMP (1)
    Header Checksum: 0xb6c4 [validation disabled]
  
```

Figura 7: Detalhes da primeira mensagem

D. O datagrama IP foi fragmentado? Justifique.

R: Não, o datagrama IP não foi fragmentado, tal como se pode verificar pelo campo das *Flags* com valor 0. Para além disto o campo *More Fragments* e *Fragment Offset* encontram-se ambos com o valor 0 e, este último é o que permite assumir se este é o primeiro fragmento ou se não foi fragmentado de todo.

```

  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: 255
    Protocol: ICMP (1)
    Header Checksum: 0x8376 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.34.32
    Destination Address: 193.136.9.254
    [3 IPv4 Fragments (3996 bytes): #3(1480), #4(1480), #5(16
  
```

Figura 8: Detalhes de fragmentação da primeira mensagem

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.

R: O valor de TTL, de *Identification* e o campo de *Checksum* vão se alterando conforme se selecionam mensagens diferentes.

2509	109.926884	172.26.254.254	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2597	110.158051	172.26.254.254	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2605	110.494121	172.26.254.254	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2613	111.785274	172.26.254.254	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2621	112.427543	172.26.254.254	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
1	0.000000	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17498/23188, ttl=1 (no response found!)
5	0.850766	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17499/23184, ttl=1 (no response found!)
5	0.181300	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17500/23628, ttl=3 (reply in 6)
7	0.344850	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17501/23876, ttl=255 (reply in 8)
9	0.395576	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17502/24132, ttl=1 (no response found!)
11	0.445619	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17503/24388, ttl=2 (no response found!)
13	0.496161	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17504/24644, ttl=3 (reply in 14)
19	1.616892	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17505/24900, ttl=255 (reply in 20)
21	1.667285	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17506/25156, ttl=1 (no response found!)
23	1.718177	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17507/25412, ttl=2 (no response found!)
25	1.768373	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17508/25668, ttl=3 (reply in 26)
27	2.247206	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17509/25924, ttl=255 (reply in 28)
29	2.297835	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17510/26180, ttl=1 (no response found!)
31	2.348455	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17511/26436, ttl=2 (no response found!)
33	2.399819	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17512/26692, ttl=3 (reply in 34)
35	2.449618	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17513/26948, ttl=255 (reply in 36)
37	2.499816	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17514/27204, ttl=1 (no response found!)
39	2.551221	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17515/27460, ttl=2 (no response found!)
41	2.601448	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17516/27716, ttl=3 (reply in 42)

Figura 9: Pacotes capturados ordenados pela *source*

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

R: Os valores do campo *Identification* são sempre somados mais 1 de cada vez que é capturada uma mensagem e o TTL também vai aumentando de 1 em 1 a cada 3 mensagens.

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?

R: Segundo a imagem abaixo, os valores do campo TTL vão se alternando entre 253 e 255.

2007	110.540390	172.16.2.1	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2009	110.597415	193.136.9.254	172.26.34.32	ICMP	70 Echo (ping) reply id=0x0001, seq=18208/8263, ttl=253 (request in 2008)
2011	111.733595	193.136.9.254	172.26.34.32	ICMP	70 Echo (ping) reply id=0x0001, seq=18209/8519, ttl=253 (request in 2010)
2013	111.785274	172.26.254.254	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2015	111.834461	172.16.2.1	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2017	111.805797	193.136.9.254	172.26.34.32	ICMP	70 Echo (ping) reply id=0x0001, seq=18212/9127, ttl=253 (request in 2016)
2019	112.377575	193.136.9.254	172.26.34.32	ICMP	70 Echo (ping) reply id=0x0001, seq=18213/9543, ttl=253 (request in 2018)
2021	112.427543	172.26.254.254	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2023	112.477327	172.16.2.1	172.26.34.32	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
2025	112.528506	193.136.9.254	172.26.34.32	ICMP	70 Echo (ping) reply id=0x0001, seq=18216/10311, ttl=253 (request in 2024)
2027	112.577713	193.136.9.254	172.26.34.32	ICMP	70 Echo (ping) reply id=0x0001, seq=18217/10567, ttl=253 (request in 2026)
1	0.000000	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17498/23188, ttl=1 (no response found!)
3	0.850766	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17499/23354, ttl=1 (no response found!)
5	0.181300	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17500/23628, ttl=3 (reply in 6)
7	0.344850	172.26.34.32	193.136.9.254	ICMP	70 Echo (ping) request id=0x0001, seq=17501/23876, ttl=255 (reply in 8)

Figura 10: Pacotes capturados ordenados pela *destination*

Exercício 3

Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempo de captura. Observe o tráfego depois do tamanho do pacote ter sido definido para (4000 + X) bytes.

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

R: Houve necessidade de fragmentar o pacote inicial porque este excede o limite de 1500 *bytes* de transmissão de uma mensagem e, neste caso, redefiniu-se o pacote para 4016 *bytes*.

6	0.288286	193.136.9.254	172.26.34.32	IPv4	1514	Fragmented IP protocol (proto-ICMP 1, off=0, ID=9933) [Reassembled in #8]
7	0.288410	193.136.9.254	172.26.34.32	IPv4	1514	Fragmented IP protocol (proto-ICMP 1, off=1488, ID=9933) [Reassembled in #8]
8	0.288485	193.136.9.254	172.26.34.32	ICMP	1870	Echo (ping) reply ID=99001, seq=29773/19828, ttl=255 [request in 5]
9	0.322666	172.26.34.32	193.136.9.254	IPv4	1514	Fragmented IP protocol (proto-ICMP 1, off=0, ID=9934) [Reassembled in #11]
10	0.322666	172.26.34.32	193.136.9.254	IPv4	1514	Fragmented IP protocol (proto-ICMP 1, off=1488, ID=9934) [Reassembled in #11]
11	0.322666	172.26.34.32	193.136.9.254	ICMP	1870	Echo (ping) request ID=99001, seq=29774/20084, ttl=1 (no response found)
12	0.351931	172.26.254.254	172.26.34.32	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)


```

0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1056
  Identification: 0x9933 (39219)
  Flags: 0x01
    0... .... = Reserved bit: Not set
    .0... .... = Don't Fragment: Not set
    ..0... .... = More Fragments: Not set
    ...0 1011 1001 0000 = Fragment Offset: 2060
  Time to Live: 253
  Protocol: ICMP (1)
  Header Checksum: 0x8576 [validation disabled]
  [Header checksum status: Unverified]
  Source Address: 193.136.9.254

```

Figura 11: Detalhes da fragmentação do pacote

- 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?

R: Pode-se verificar que o datagrama IP foi fragmentado pelo campo das *Flags* com valor 1. Trata-se do primeiro fragmento pois *Fragment Offset* tem valor 0 e como o campo *More Fragments* encontra-se a 1 é porque existem mais fragmentos para além deste. O tamanho deste datagrama IP é de 1500 *bytes*.

```

Total Length: 1500
Identification: 0x9934 (39220)
▼ 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

```

Figura 12: Detalhes 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?

R: Como este fragmento tem o mesmo *Identification* do anterior, confirma-se que é o segundo fragmento do datagrama IP original. O campo *Fragment Offset* indica que não se trata do primeiro fragmento, é diferente de 0. O campo *More Fragments* é 1 logo, existem mais fragmentos.

```

Total Length: 1500
Identification: 0x9934 (39220)
▼ 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

```

Figura 13: Detalhes do segundo fragmento

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

R: A partir do datagrama original foram criados 3 fragmentos.

```

▼ [3 IPv4 Fragments (3996 bytes): #6(1480), #7(1480), #8(1036)]
[Frame: 6, payload: 0-1479 (1480 bytes)]
[Frame: 7, payload: 1480-2959 (1480 bytes)]
[Frame: 8, payload: 2960-3995 (1036 bytes)]
[Fragment count: 3]

```

Figura 14: Fragmentos criados

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.

R: Os campos que se alteram entre os diferentes fragmentos são o *Fragment Offset*, *Total Length* e, também, a *Header Checksum*. Como todos os fragmentos provenientes do datagrama original tem o mesmo *Identification* podemos reconstruir o mesmo. Para ser possível saber a ordem dos fragmentos basta ter em atenção ao campo *Fragment Offset* e ao *Total Length* de modo a alocar o espaço corretamente.

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

R:

$Length = 4016 \text{ bytes}$
 $4016 - 20 = 3996 \text{ bytes}$ (20 bytes de cabeçalho)
 $MTU = 1500 \text{ bytes}$
 $Identification = 0x9934 = 39220$
 $Fragmentos = 1500 - 20 = 1480 \text{ bytes dos dados}$

Fragmento 1

campo de dados = 3996 bytes
 $More\ Fragments = 1$
 $Fragment\ offset = 0$
 $3996 - 1480 = 2516 \text{ bytes}$

Fragmento 2

campo de dados= 2516 bytes

$2516 - 1480 = 1036$ bytes

More Fragments = 1

Fragment offset = 1480 bytes

Fragmento 3

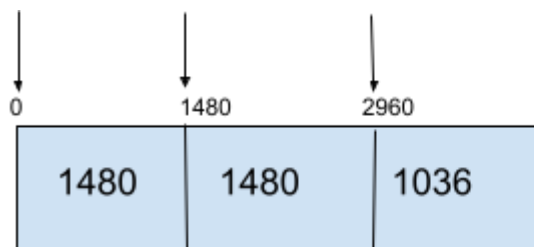
campo de dados= 1036 bytes

More Fragments = 0

Fragment offset = 2960

1 pacote = 4016 bytes

3 pacotes = $4016 + 3 \times 20 = 4076$



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

R:

$\text{MoreFragments} == 0 \ \&\& (\text{offset} + \text{Tamanho Fragmento} > \text{length}) \ \&\& \text{identification} == 39220$

De forma a detetar o último fragmento temos que ter em atenção ao campo *Fragment Offset*, este valor deve ser 0. O campo *Identification* também deve ser o mesmo em todos os fragmentos, inclusive no último.

2ª parte

Exercício 1

Atenda aos endereços IP atribuídos automaticamente pelo CORE aos diversos equipamentos da topologia.

A. Indique que endereços IP e máscaras de rede foram atribuídos pelo CORE a cada equipamento. Para simplificar, pode incluir uma imagem que ilustra de forma clara a topologia definida e o endereçamento usado.

R: Os endereços IP são referidos na imagem seguinte e as máscaras de redes tem 24 bits.

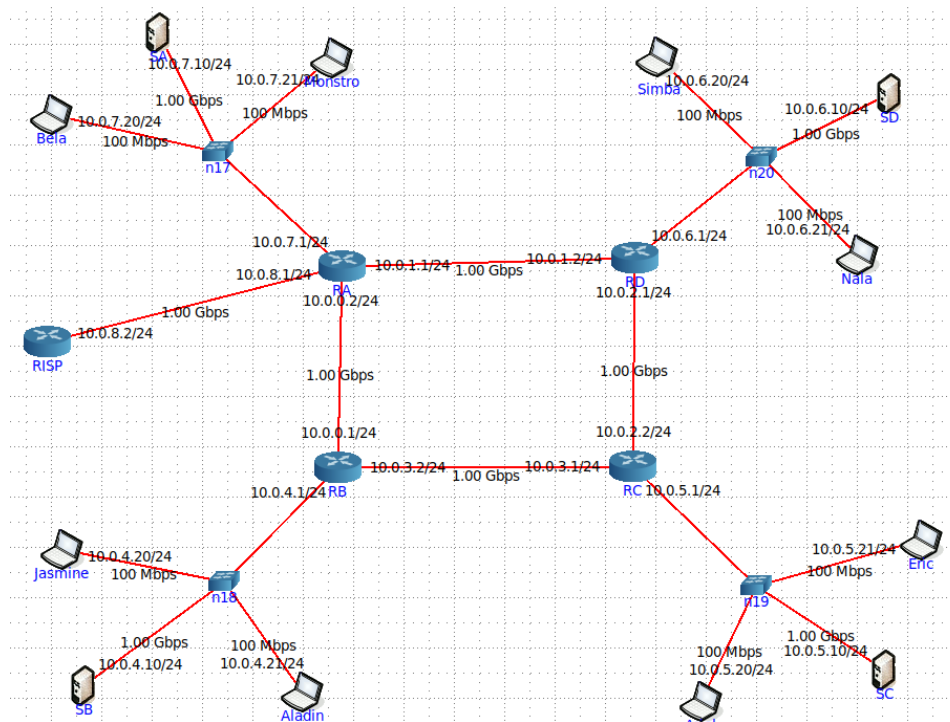


Figura 15: Core de acordo com a topologia

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

R: Estão no range dos endereços privados, logo são endereços privados.

C. Por que razão não é atribuído um endereço IP aos switches?

R: Os *switches* são dispositivos que se conectam com todos os elementos de uma rede, operando como ligação de dados, por tanto não precisam de adquirir um endereço IP pois a rede que eles propagam já contém um endereço.

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

R: Depois de efetuarmos o comando ping para um dos computadores dentro do mesmo departamento, confirmou-se que existe conectividade entre os dois, tal como podemos observar na figura 16.

```

root@SC:/tmp/pycore.36577/SC.conf# ping 10.0.5.21
PING 10.0.5.21 (10.0.5.21) 56(84) bytes of data.
64 bytes from 10.0.5.21: icmp_seq=1 ttl=64 time=0,831 ms
64 bytes from 10.0.5.21: icmp_seq=2 ttl=64 time=0,146 ms
64 bytes from 10.0.5.21: icmp_seq=3 ttl=64 time=0,135 ms
^C
--- 10.0.5.21 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2010ms
rtt min/avg/max/mdev = 0,135/0,370/0,831/0,325 ms
root@SC:/tmp/pycore.36577/SC.conf# ping 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=1,07 ms
64 bytes from 10.0.5.20: icmp_seq=2 ttl=64 time=0,168 ms
64 bytes from 10.0.5.20: icmp_seq=3 ttl=64 time=0,247 ms
64 bytes from 10.0.5.20: icmp_seq=4 ttl=64 time=0,248 ms
^C
--- 10.0.5.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3034ms
rtt min/avg/max/mdev = 0,168/0,433/1,072/0,369 ms
root@SC:/tmp/pycore.36577/SC.conf# █

```

Figura 16: Comando ping na shell do servidor do departamento C, nele mesmo

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

R: Depois de efetuarmos o comando ping para um dos computadores de cada um dos departamentos, confirmou-se que existe conectividade entre os dois, tal como podemos observar na figura abaixo.

```

root@SA:/tmp/pycore.36577/SA.conf# ping 10.0.7.21
PING 10.0.7.21 (10.0.7.21) 56(84) bytes of data.
64 bytes from 10.0.7.21: icmp_seq=1 ttl=64 time=1,23 ms
64 bytes from 10.0.7.21: icmp_seq=2 ttl=64 time=0,103 ms
64 bytes from 10.0.7.21: icmp_seq=3 ttl=64 time=0,157 ms
64 bytes from 10.0.7.21: icmp_seq=4 ttl=64 time=0,640 ms
^C
--- 10.0.7.21 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3040ms
rtt min/avg/max/mdev = 0,103/0,531/1,226/0,452 ms
root@SA:/tmp/pycore.36577/SA.conf# ping 10.0.4.20
PING 10.0.4.20 (10.0.4.20) 56(84) bytes of data.
64 bytes from 10.0.4.20: icmp_seq=1 ttl=62 time=2,30 ms
64 bytes from 10.0.4.20: icmp_seq=2 ttl=62 time=0,453 ms
64 bytes from 10.0.4.20: icmp_seq=3 ttl=62 time=0,777 ms
^C
--- 10.0.4.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0,453/1,176/2,299/0,804 ms
root@SA:/tmp/pycore.36577/SA.conf# ping 10.0.6.20
PING 10.0.6.20 (10.0.6.20) 56(84) bytes of data.
64 bytes from 10.0.6.20: icmp_seq=1 ttl=62 time=0,893 ms
64 bytes from 10.0.6.20: icmp_seq=2 ttl=62 time=0,429 ms
64 bytes from 10.0.6.20: icmp_seq=3 ttl=62 time=1,19 ms
^C
--- 10.0.6.20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2007ms
rtt min/avg/max/mdev = 0,429/0,835/1,185/0,311 ms
root@SA:/tmp/pycore.36577/SA.conf# ping 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=61 time=0,381 ms
64 bytes from 10.0.5.20: icmp_seq=2 ttl=61 time=0,337 ms
64 bytes from 10.0.5.20: icmp_seq=3 ttl=61 time=0,411 ms
64 bytes from 10.0.5.20: icmp_seq=4 ttl=61 time=0,711 ms
^C
--- 10.0.5.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3045ms
rtt min/avg/max/mdev = 0,337/0,460/0,711/0,147 ms
root@SA:/tmp/pycore.36577/SA.conf# █

```

Figura 17: Comando ping na shell do servidor do departamento A, entre outros

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

R: Através do comando ping é possível verificar se existe conectividade entre o portátil Bela e o router de acesso RISP. Tal como se pode visualizar na imagem, foram enviados 5 pacotes e recebidos do mesmo valor, ou seja, há conectividade entre eles.

```
root@Bela:/tmp/pycore.36577/Bela.conf# ping 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=0.966 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=63 time=0.465 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=63 time=0.234 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=63 time=0.316 ms
64 bytes from 10.0.8.2: icmp_seq=5 ttl=63 time=0.391 ms
^C
--- 10.0.8.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4070ms
rtt min/avg/max/mdev = 0.234/0.474/0.966/0.257 ms
root@Bela:/tmp/pycore.36577/Bela.conf#
```

Figura 18: Comando ping entre o portátil Bela e router RISP

Exercício 2

Para o router RA e o portátil Bela:

- 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 respectivo (`man netstat`).

R: A primeira coluna, *Destination*, da tabela de encaminhamento refere-se aos endereços IP destino, especificando o destino da conexão. Na segunda coluna podemos ver o *Gateway* utilizado pela conexão. A coluna *Genmask* mostra a máscara de rede utilizada. A coluna *Flags* mostra as *flags* da conexão. A coluna *MSS* representa o *Maximum Segment Size* desta rota. *Window* especifica o tamanho da janela para conexões TCP nesta *route*. Por fim, a última coluna diz-nos qual é a interface que está a ser utilizada no encaminhamento desta conexão.

```
root@Bela:/tmp/pycore.36577/Bela.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.7.1 0.0.0.0 UG 0 0 0 eth0
10.0.7.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@Bela:/tmp/pycore.36577/Bela.conf#
```

Figura 19: Tabela de encaminhamento do portátil Bela

```

root@RA:/tmp/pycore.36577/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            10.0.1.2          255.255.255.0    UG       0 0        0 eth1
10.0.3.0            10.0.0.1          255.255.255.0    UG       0 0        0 eth0
10.0.4.0            10.0.0.1          255.255.255.0    UG       0 0        0 eth0
10.0.5.0            10.0.0.1          255.255.255.0    UG       0 0        0 eth0
10.0.6.0            10.0.1.2          255.255.255.0    UG       0 0        0 eth1
10.0.7.0            0.0.0.0           255.255.255.0    U        0 0        0 eth2
10.0.8.0            0.0.0.0           255.255.255.0    U        0 0        0 eth3
root@RA:/tmp/pycore.36577/RA.conf#

```

Figura 20: Tabela de encaminhamento do router A

- 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).

R: De acordo com a figura abaixo, podemos observar que o protocolo OSPF está ativo, logo está a ser utilizado um encaminhamento dinâmico.

```

root@Bela:/tmp/pycore.36577/Bela.conf# ps -ax
  PID TTY          STAT       TIME COMMAND
    1 ?            S          0:00 vncnode -v -c /tmp/pycore.36577/Bela -l /tmp/pycore.
   36 pts/3      Ss         0:00 /bin/bash
   43 pts/3      R+         0:00 ps -ax
root@Bela:/tmp/pycore.36577/Bela.conf#

```

Figura 21: Comando ps-ax do portátil Bela

```

root@RA:/tmp/pycore.36577/RA.conf# ps -ax
  PID TTY          STAT       TIME COMMAND
    1 ?            S          0:00 vncnode -v -c /tmp/pycore.36577/RA -l /tmp/pycore.36577/RA.log -p /tmp/pycore.36577/RA.pid -C /tmp/pycore.36577/RA.conf
   75 ?            Ss         0:00 /usr/local/sbin/zebra -d
   81 ?            Ss         0:00 /usr/local/sbin/ospfd -d
   85 ?            Ss         0:00 /usr/local/sbin/ospfd -d
   93 pts/3        Ss         0:00 /bin/bash
  101 pts/3        R+         0:00 ps -ax
root@RA:/tmp/pycore.36577/RA.conf#

```

Figura 22: Comando ps-ax do router A

- 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.

R: Se a rota por defeito for retirada definitivamente, o servidor de um certo departamento apenas consegue uma conexão com outros da mesma sub-rede. Tal como podemos observar na figura abaixo, não existe comunicação entre o servidor do departamento A e um host de uma qualquer outra sub-rede.

```

root@SA:/tmp/pycore.36577/SA.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0          10.0.7.1       0.0.0.0         UG        0 0        0 eth0
10.0.7.0         0.0.0.0        255.255.255.0   U        0 0        0 eth0
root@SA:/tmp/pycore.36577/SA.conf# route delete default
root@SA:/tmp/pycore.36577/SA.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.7.0         0.0.0.0        255.255.255.0   U        0 0        0 eth0
root@SA:/tmp/pycore.36577/SA.conf#

```

Figura 23: Tabela de encaminhamento do servidor A sem a rota por defeito

- 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 registe os comandos que usou.

R:

```

root@SA:/tmp/pycore.36521/SA.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.7.1
root@SA:/tmp/pycore.36521/SA.conf# route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.7.1
root@SA:/tmp/pycore.36521/SA.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.7.1
root@SA:/tmp/pycore.36521/SA.conf# route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.7.1
root@SA:/tmp/pycore.36521/SA.conf# netstat -rn
Kernel IP routing table
Destination      Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.4.0         10.0.7.1       255.255.255.0   UG        0 0        0 eth0
10.0.5.0         10.0.7.1       255.255.255.0   UG        0 0        0 eth0
10.0.6.0         10.0.7.1       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 eth0
10.0.8.0         10.0.7.1       255.255.255.0   UG        0 0        0 eth0

```

Figura 24: Adição de rotas necessárias

- 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.

R:

```
root@SA:/tmp/pycore.36521/SA.conf# ping 10.0.6,20
PING 10.0.6,20 (10.0.6.20) 56(84) bytes of data.
64 bytes from 10.0.6,20: icmp_seq=1 ttl=62 time=1,29 ms
64 bytes from 10.0.6,20: icmp_seq=2 ttl=62 time=0,438 ms
64 bytes from 10.0.6,20: icmp_seq=3 ttl=62 time=0,906 ms
64 bytes from 10.0.6,20: icmp_seq=4 ttl=62 time=0,458 ms
^C
--- 10.0.6,20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3020ms
rtt min/avg/max/mdev = 0.438/0.772/1.288/0.351 ms
root@SA:/tmp/pycore.36521/SA.conf# ping 10.0,4,20
PING 10.0,4,20 (10.0.4,20) 56(84) bytes of data.
64 bytes from 10.0,4,20: icmp_seq=1 ttl=62 time=1,32 ms
64 bytes from 10.0,4,20: icmp_seq=2 ttl=62 time=0,310 ms
64 bytes from 10.0,4,20: icmp_seq=3 ttl=62 time=0,608 ms
^C
--- 10.0,4,20 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2019ms
rtt min/avg/max/mdev = 0.310/0.745/1.318/0.422 ms
root@SA:/tmp/pycore.36521/SA.conf# ping 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=61 time=9,69 ms
64 bytes from 10.0,5,20: icmp_seq=2 ttl=61 time=4,37 ms
64 bytes from 10.0,5,20: icmp_seq=3 ttl=61 time=3,53 ms
64 bytes from 10.0,5,20: icmp_seq=4 ttl=61 time=4,06 ms
^C
--- 10.0,5,20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 3.532/5.414/9.694/2.489 ms
root@SA:/tmp/pycore.36521/SA.conf# ping 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=1,45 ms
64 bytes from 10.0,8,2: icmp_seq=2 ttl=63 time=0,619 ms
64 bytes from 10.0,8,2: icmp_seq=3 ttl=63 time=0,584 ms
^C
--- 10.0,8,2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2023ms
rtt min/avg/max/mdev = 0.584/0.884/1.449/0.399 ms
root@SA:/tmp/pycore.36521/SA.conf#
```

Figura 25: Comando ping na shell do servidor do departamento A, entre outros

Exercício 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 endereçamento das redes em questão.

Para definir endereços de sub-rede é necessário usar a parte prevista para endereçamento de host, não sendo possível alterar o endereço de rede original. Recordar-se que o subnetting, ao recorrer ao espaço de endereçamento para host, implica que possam ser endereçados menos hosts.

Considere a topologia definida anteriormente. Assuma que o endereçamento entre os routers (rede de backbone) se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.

1. Considere que dispõe apenas do endereço de rede IP 192.168.XXX.128/25, em que XXX é o decimal correspondente 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.

R: Usamos 4 sub-redes logo vamos precisar de 2 bits para a representar (em binário). Ficamos então com uma nova máscara de 29. Depois disso mudamos o endereço IP em todos os departamentos usando a nova máscara. Na figura abaixo vemos a nova topologia com 4 sub-redes.

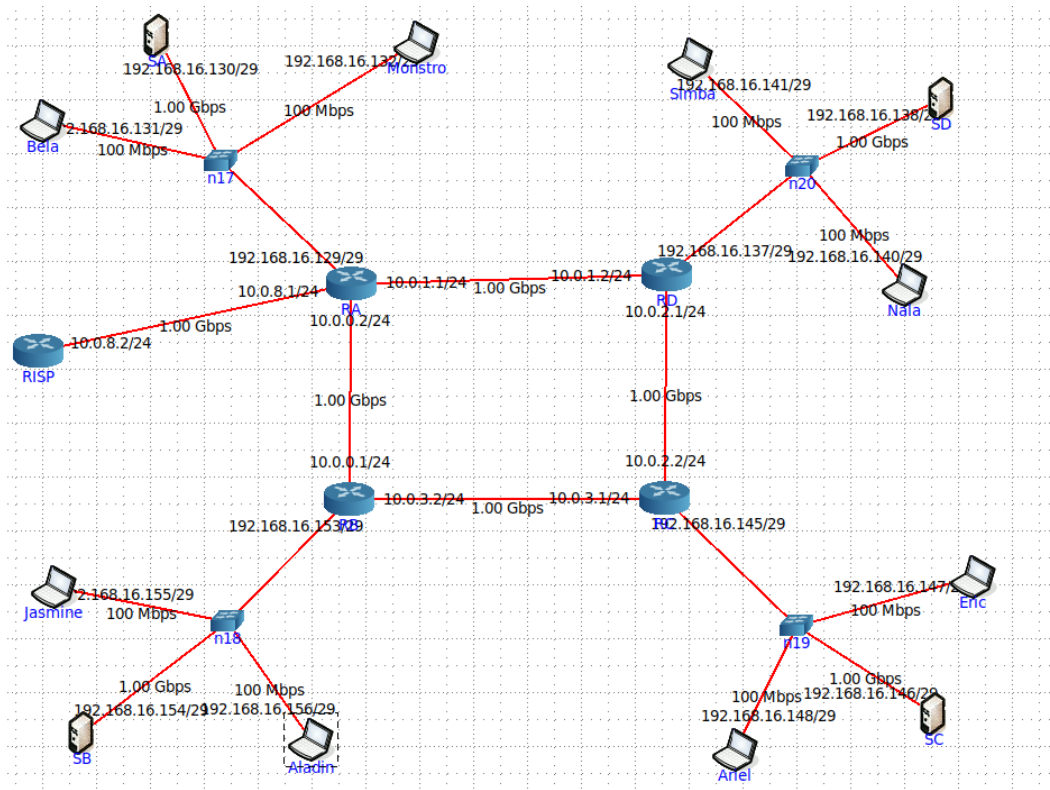


Figura 26: Core de acordo com as novas alterações na topologia

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

R: Foi utilizada máscara de rede /29 que em formato decimal é 255.255.255.248. O número de hosts possíveis é 6 ($2^3 - 2 = 6$), porque há 3 bits para host e dois deles são reservados. Já a nível de sub-redes há 16 ($2^4 = 16$) disponíveis, pois são usados 4 bits para identificar as mesmas.

3. 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.

R: Tal como se pode observar nas imagens seguintes, há conectividade entre os computadores dos diversos departamentos e o *router* RISP. Por exemplo, há conexão entre o *host* Aladin do departamento B e o *router* RISP, entre outros.

```

root@Aladin:/tmp/pycore.35325/Aladin.conf# ping 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=62 time=0,867 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=62 time=0,794 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=62 time=0,992 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=62 time=1,87 ms
64 bytes from 10.0.8.2: icmp_seq=5 ttl=62 time=1,14 ms
64 bytes from 10.0.8.2: icmp_seq=6 ttl=62 time=0,619 ms
^C
--- 10.0.8.2 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5009ms
rtt min/avg/max/mdev = 0,619/1,047/1,870/0,401 ms
root@Aladin:/tmp/pycore.35325/Aladin.conf# █

```

Figura 27: Conectividade entre computador Aladin e router RISP

```

root@Ariel:/tmp/pycore.35325/Ariel.conf# ping 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=61 time=1,02 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=61 time=0,881 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=61 time=0,796 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=61 time=1,70 ms
64 bytes from 10.0.8.2: icmp_seq=5 ttl=61 time=1,14 ms
^C
--- 10.0.8.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4049ms
rtt min/avg/max/mdev = 0,796/1,106/1,695/0,316 ms
root@Ariel:/tmp/pycore.35325/Ariel.conf# █

```

Figura 28: Conectividade entre computador Ariel e router RISP

```

root@Bela:/tmp/pycore.35325/Bela.conf# ping 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=0,711 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=63 time=0,749 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=63 time=0,263 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=63 time=0,201 ms
^C
--- 10.0.8.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 0,201/0,481/0,749/0,250 ms
root@Bela:/tmp/pycore.35325/Bela.conf# █

```

Figura 29: Conectividade entre computador Bela e router RISP

```

root@Simba:/tmp/pycore.35325/Simba.conf# ping 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=62 time=2,53 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=62 time=1,02 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=62 time=0,775 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=62 time=1,11 ms
^C
--- 10.0.8.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3014ms
rtt min/avg/max/mdev = 0,775/1,358/2,528/0,686 ms
root@Simba:/tmp/pycore.35325/Simba.conf# █

```

Figura 30: Conectividade entre computador Simba e router RISP

Já nas imagens que se seguem pode-se verificar que existe conectividade entre os *routers* dos diversos departamentos. Por exemplo, há conexão entre o *router A* e o *router D*, entre outros.

```
root@RA:/tmp/pycore.35325/RA.conf# ping 192.168.16.137
PING 192.168.16.137 (192.168.16.137) 56(84) bytes of data.
64 bytes from 192.168.16.137: icmp_seq=1 ttl=64 time=0.361 ms
64 bytes from 192.168.16.137: icmp_seq=2 ttl=64 time=0.639 ms
64 bytes from 192.168.16.137: icmp_seq=3 ttl=64 time=0.360 ms
64 bytes from 192.168.16.137: icmp_seq=4 ttl=64 time=0.364 ms
^C
--- 192.168.16.137 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3065ms
rtt min/avg/max/mdev = 0.360/0.431/0.639/0.120 ms
root@RA:/tmp/pycore.35325/RA.conf#
```

Figura 31: Conectividade entre router A e router D

```
root@RB:/tmp/pycore.35325/RB.conf# ping 192.168.16.129
PING 192.168.16.129 (192.168.16.129) 56(84) bytes of data.
64 bytes from 192.168.16.129: icmp_seq=1 ttl=64 time=1.31 ms
64 bytes from 192.168.16.129: icmp_seq=2 ttl=64 time=0.324 ms
64 bytes from 192.168.16.129: icmp_seq=3 ttl=64 time=0.282 ms
64 bytes from 192.168.16.129: icmp_seq=4 ttl=64 time=0.248 ms
^C
--- 192.168.16.129 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3030ms
rtt min/avg/max/mdev = 0.248/0.541/1.312/0.445 ms
root@RB:/tmp/pycore.35325/RB.conf#
```

Figura 32: Conectividade entre router B e router A

```
root@RC:/tmp/pycore.35325/RC.conf# ping 192.168.16.153
PING 192.168.16.153 (192.168.16.153) 56(84) bytes of data.
64 bytes from 192.168.16.153: icmp_seq=1 ttl=64 time=0.600 ms
64 bytes from 192.168.16.153: icmp_seq=2 ttl=64 time=0.166 ms
64 bytes from 192.168.16.153: icmp_seq=3 ttl=64 time=0.308 ms
^C
--- 192.168.16.153 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2035ms
rtt min/avg/max/mdev = 0.166/0.358/0.600/0.180 ms
root@RC:/tmp/pycore.35325/RC.conf#
```

Figura 33: Conectividade entre router C e router B

```
root@RD:/tmp/pycore.35325/RD.conf# ping 192.168.16.145
PING 192.168.16.145 (192.168.16.145) 56(84) bytes of data.
64 bytes from 192.168.16.145: icmp_seq=1 ttl=64 time=0.893 ms
64 bytes from 192.168.16.145: icmp_seq=2 ttl=64 time=0.169 ms
64 bytes from 192.168.16.145: icmp_seq=3 ttl=64 time=0.460 ms
64 bytes from 192.168.16.145: icmp_seq=4 ttl=64 time=0.176 ms
^C
--- 192.168.16.145 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3064ms
rtt min/avg/max/mdev = 0.169/0.424/0.893/0.294 ms
root@RD:/tmp/pycore.35325/RD.conf#
```

Figura 34: Conectividade entre router D e router C

Conclusão

Neste trabalho aprendemos a usar softwares de controle e administração de redes, como o Wireshark e o Core. Fundamentamos conceitos importantes, que são um objetivo final da aprendizagem desta unidade curricular como o processo de fragmentação, o endereçamento, sub-redes entre outros. Tivemos algumas dificuldades no utilização do Core mas com algum esforço conseguimos superar os percalços. Acreditamos que este trabalho foi relevante para nos tornarmos melhores profissionais e pessoas mais cultas.