Universidade do Minho

Licenciatura em Engenharia Informática

Redes de Computadores

Grupo 52

TP2 : Protocolo IPv4

António Luís Braga Mendes (A84675)

Maria Eugénia Bessa Cunha (A93264)

Vicente Gonçalves Moreira (A93296)

Março 2022

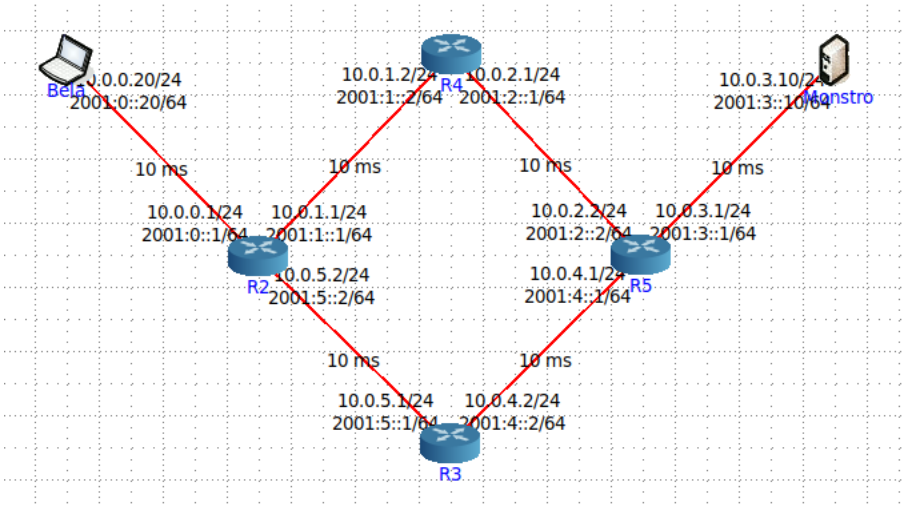
1 Parte 1 - Questões e Respostas

1.1 Questão 1 - Traceroute na topologia CORE

b. Active o wireshark ou o tcpdump no host Bela. Numa shell de Bela execute o comando traceroute -I para o endereço IP do Monstro. Registe e analise o tráfego ICMP enviado pelo sistema Bela e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

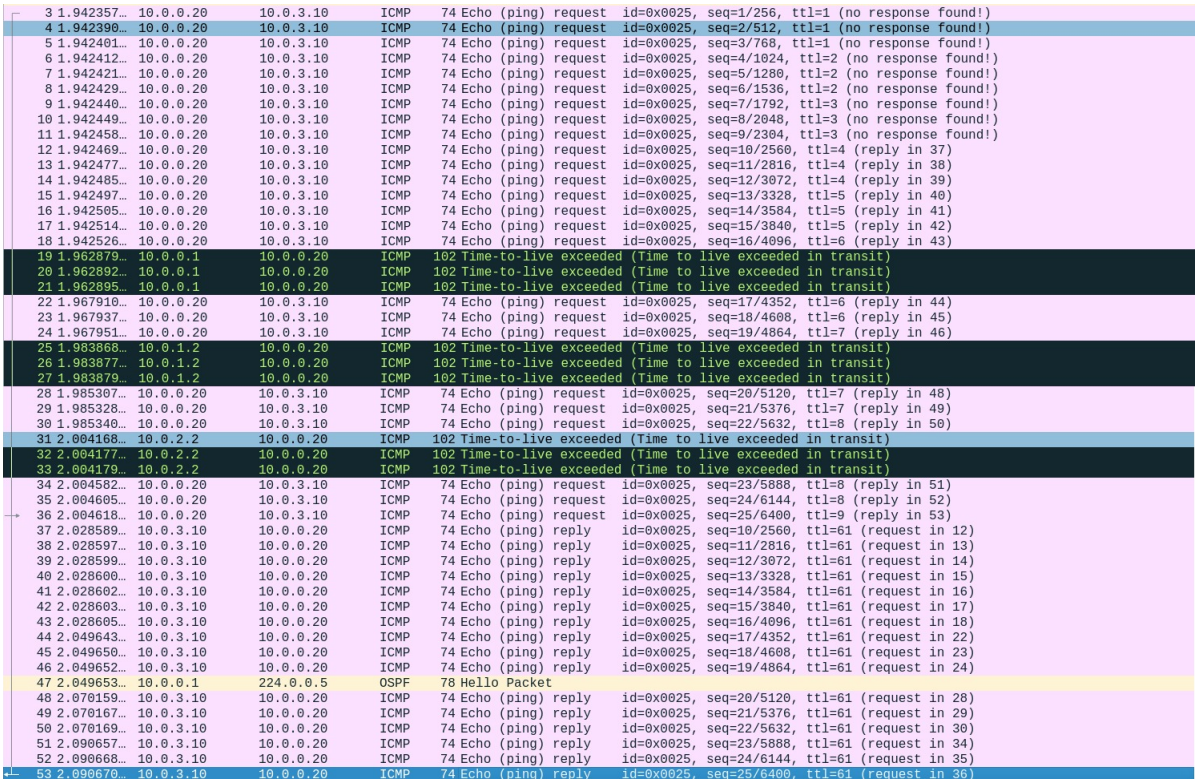
Começamos por criar a topologia da rede no ambiente *CORE*, seguindo as especificações que nos foram pedidas.

Figura 1: Topologia CORE



De seguida, executamos na *shell* do *host* "Bela" o comando *traceroute -I 10.0.3.10* e capturamos os datagramas, utilizando o programa *Wireshark*, sendo esta captura a seguinte:

Figura 2: Captura *TraceRoute* Bela-Monstro



3	1.942357…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=1/256, ttl=1 (no response found!)
4	1.942390…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=2/512, ttl=1 (no response found!)
5	1.942401…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=3/768, ttl=1 (no response found!)
6	1.942412…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=4/1024, ttl=2 (no response found!)
7	1.942421…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=5/1280, ttl=2 (no response found!)
8	1.942429…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=6/1536, ttl=2 (no response found!)
9	1.942440…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=7/1792, ttl=3 (no response found!)
10	1.942449…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=8/2048, ttl=3 (no response found!)
11	1.942458…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=9/2304, ttl=3 (no response found!)
12	1.942469…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=10/2560, ttl=4 (reply in 37)
13	1.942477…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=11/2816, ttl=4 (reply in 38)
14	1.942485…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=12/3072, ttl=4 (reply in 39)
15	1.942497…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=13/3328, ttl=5 (reply in 40)
16	1.942505…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=14/3584, ttl=5 (reply in 41)
17	1.942514…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=15/3840, ttl=5 (reply in 42)
18	1.942526…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=16/4096, ttl=6 (reply in 43)
19	1.962879…	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
20	1.962892…	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
21	1.962895…	10.0.0.1	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
22	1.967910…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=17/4352, ttl=6 (reply in 44)
23	1.967937…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=18/4608, ttl=6 (reply in 45)
24	1.967951…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=19/4864, ttl=7 (reply in 46)
25	1.983868…	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
26	1.983877…	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
27	1.983879…	10.0.1.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
28	1.985307…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=20/5120, ttl=7 (reply in 48)
29	1.985328…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=21/5376, ttl=7 (reply in 49)
30	1.985340…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=22/5632, ttl=8 (reply in 50)
31	2.004168…	10.0.2.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
32	2.004177…	10.0.2.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
33	2.004179…	10.0.2.2	10.0.0.20	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
34	2.004582…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=23/5888, ttl=8 (reply in 51)
35	2.004605…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=24/6144, ttl=8 (reply in 52)
36	2.004618…	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0025, seq=25/6400, ttl=9 (reply in 53)
37	2.028589…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=10/2560, ttl=61 (request in 12)
38	2.028597…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=11/2816, ttl=61 (request in 13)
39	2.028599…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=12/3072, ttl=61 (request in 14)
40	2.028600…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=13/3328, ttl=61 (request in 15)
41	2.028602…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=14/3584, ttl=61 (request in 16)
42	2.028603…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=15/3840, ttl=61 (request in 17)
43	2.028605…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=16/4096, ttl=61 (request in 18)
44	2.049643…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=17/4352, ttl=61 (request in 22)
45	2.049650…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=18/4608, ttl=61 (request in 23)
46	2.049652…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=19/4864, ttl=61 (request in 24)
47	2.049653…	10.0.0.1	224.0.0.5	OSPF	78	Hello Packet
48	2.070159…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=20/5120, ttl=61 (request in 28)
49	2.070167…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=21/5376, ttl=61 (request in 29)
50	2.070169…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=22/5632, ttl=61 (request in 30)
51	2.090657…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=23/5888, ttl=61 (request in 34)
52	2.090668…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=24/6144, ttl=61 (request in 35)
53	2.090670…	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0025, seq=25/6400, ttl=61 (request in 36)

Como podemos observar a partir do campo TTL (*time to live*), verificamos que o comando *traceroute* tenta alcançar o IP desejado em 3 tentativas distintas e, sempre que estas não recebam a resposta do IP destino, o TTL é incrementado e o processo é repetido, até que o IP seja alcançado. Este comando é útil para traçar o caminho na rede que os datagramas seguem para alcançar um determinado IP na rede, pois por cada tentativa falhada (quando o TTL chega a 0), o *router*, por defeito, sinaliza o *host* de origem que o seu datagrama não foi entregue, o que nos permite registar o IP desse mesmo *router*. Ao efetuar envios com TTL's crescentes podemos assim registar todos os endereços IP utilizados no envio da mensagem ao destino.

Neste exemplo, podemos observar que as tentativas de comunicação com os TTL igual a 1,2 e 3 (datagramas 3 a 11) não obtiveram resposta do IP destino e, depois de um atraso, foi recebido um aviso dos *routers* intervenientes. Ao analisar o campo de origem destes datagramas, vemos que os *routers* R2,R4 e R5 (IP's 10.0.0.1, 10.0.1.2 e 10.0.2.2, respetivamente) são os *routers* utilizados na troca de pacotes entre o *host* "Bela" e o servidor "Monstro".

Também podemos observar que, devido aos atrasos adicionados na topologia da rede, os pacotes de *request* enviados pelo *host* "Bela", apesar de conterem um TTL suficiente para alcançar o servidor "Monstro", foram considerados "não respondidos" devido aos seus atrasos, o que faz com o que o *traceroute* incremente o seu TTL e tente novamente. No nosso exemplo, este processo aconteceu várias vezes até que, depois de enviar um datagrama com um valor de TTL igual a 9, as respostas aos primeiros pedidos retornaram, o que fez o comando *traceroute* ficar à escuta destes. (16 pacotes ao todo)

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.

Visto que, na topologia originada para este exercício, temos presente no mínimo 4 saltos entre o *host* "Bela" e o servidor "Monstro", então designamos que o valor inicial mínimo do campo TTL terá que ser 4. Esta decisão provou-se correta quando ao analisar o resultado obtido utilizando o comando *traceroute -f 4 -m 4 -I 10.0.3.10* (TTL minimo de 4 + TTL máximo de 4) verificamos pela captura de pacotes que este alcançou com sucesso o IP de destino.

Figura 3: Comunicação com servidor Monstro com TTL limitado a 4

2	1.246786564	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0027, seq=1/256, ttl=4 (reply in 5)
3	1.246817970	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0027, seq=2/512, ttl=4 (reply in 6)
4	1.246828245	10.0.0.20	10.0.3.10	ICMP	74	Echo (ping) request id=0x0027, seq=3/768, ttl=4 (reply in 7)
5	1.328529263	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0027, seq=1/256, ttl=61 (request in 2)
6	1.328537762	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0027, seq=2/512, ttl=61 (request in 3)
7	1.328539574	10.0.3.10	10.0.0.20	ICMP	74	Echo (ping) reply id=0x0027, seq=3/768, ttl=61 (request in 4)

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.

Para a realização deste exercício executamos o comando *traceroute -q 10 -I 10.0.3.10* e depois de obter os resultados, calculamos a média. Neste exercício calculamos um atraso médio de 81,6 millisegundos.

Figura 4: Cálculo da médio de atraso

```
root@Bela:/tmp/pycore.44625/Bela.conf# traceroute -q 10 -I 10.0.3.10
traceroute to 10.0.3.10 (10.0.3.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  20.477 ms  20.436 ms  20.428 ms  20.420 ms  20.415 ms  20.409 ms * * * *
 2  10.0.1.2 (10.0.1.2)  40.876 ms  40.871 ms  40.866 ms  40.859 ms  40.854 ms  40.849 ms * * * *
 3  10.0.2.2 (10.0.2.2)  63.342 ms  63.337 ms  61.266 ms  61.234 ms  61.226 ms  61.220 ms * * * *
 4  10.0.3.10 (10.0.3.10)  81.623 ms  81.618 ms  81.390 ms  81.353 ms  81.770 ms  81.734 ms  81.726 ms  81.720 ms  81.564 ms  81.526 ms
root@Bela:/tmp/pycore.44625/Bela.conf#
```

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?

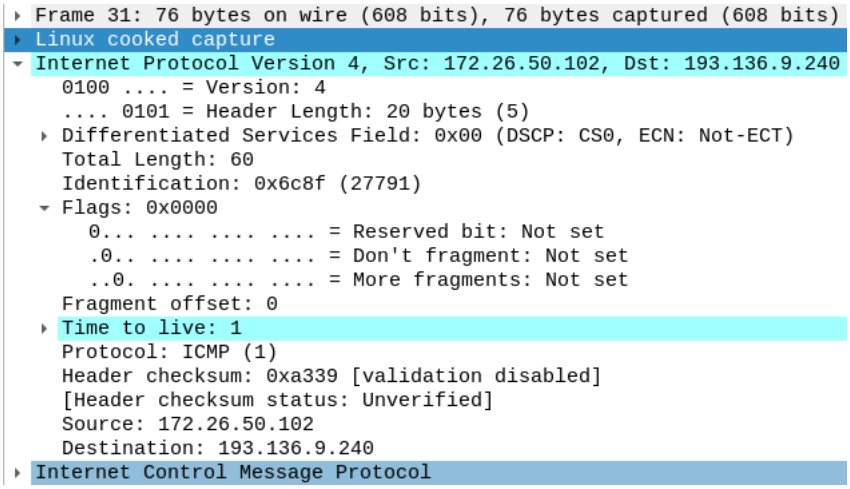
Não, dado que o valor de RTT calculado não contém informação sobre os caminhos e atrasos que os pacotes sofreram tanto no seu envio como na sua receção, fazendo com que a divisão em 2 seja uma aproximação errada e imprecisa, visto que o tempo de envio e receção pode variar significativamente.

A métrica de *One Way Delay* é difícil de calcular devido à falta de métodos robustos que sejam capazes de calcular os atrasos entre dois *hosts*. Por exemplo, pode ser utilizado uma *TimeStamp* no envio de um pacote para depois, o recetor, poder calcular o *One Way Delay*, no entanto, este método só funciona caso ambos os *hosts* tiverem os seus relógios sincronizados, algo difícil de garantir, podendo haver assim *One Way Delays* maiores ou menores do que o seu valor real.

1.2 Questão 2 - Análise do cabeçalho Ipv4

Selecionando o primeiro pacote ICMP capturado referente a situação i), obtivemos os seguintes dados:

Figura 5: Análise de Pacote ICMP sem fragmentação (i)



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

Observando o campo *Source*, concluímos que o endereço IP da máquina utilizada é 172.26.50.102

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

Observando o campo *Protocol*, este possuí o valor 1 que será correspondente ao protocolo ICMP. Este valor permite identificar o protocolo da mensagem transmitida no datagrama, permitindo que a leitura do pacote seja efetuada com sucesso.

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

Observando o campo *Header Length*, verificamos que este tem um tamanho de 20 *bytes*. Para calcular o tamanho do *payload* contido neste datagrama podemos subtrair ao tamanho total do datagrama, visto no campo *Total Length* (60 bytes), o valor do tamanho do *header*. Neste exemplo, o *payload* tem um tamanho de 40 *bytes*.

d. O datagrama IP foi fragmentado? Justifique.

Ao analisar os campos de *Flags*, em específico, as *flags Don't fragment* e *More fragments* podemos concluir que este datagrama não foi fragmentado, visto que estas flags foram definidas como *Not Set*. Devido ao tamanho reduzido do datagrama, não houve necessidade de fragmenta-lo, pois este processo é ineficiente, acrescentando novos overheads.

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.

Analisando os vários pacotes enviados a partir da nossa máquina (IP 172.26.50.102), observamos que apenas 3 campos variam ao longo dos vários envios: O campo *Identification*, dado que contêm o número de sequência do pacote, o campo *Time to Live*, visto que o comando *traceroute* vai incrementando o valor deste ao longo de várias tentativas e por último o campo *Header Checksum* pois com a mudança de alguns dados do pacote, a sua *checksum* irá variar.

Figura 6: Análise das diferenças entre pacotes enviados pela máquina 172.26.50.102

31	1.3385557…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=1/256, ttl=1 (no response found!)
32	1.3385892…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=2/512, ttl=1 (no response found!)
33	1.3385989…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=3/768, ttl=1 (no response found!)
34	1.3386094…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=4/1024, ttl=2 (no response found!)
35	1.3386223…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=5/1280, ttl=2 (no response found!)
36	1.3386335…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=6/1536, ttl=2 (no response found!)
37	1.3386470…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=7/1792, ttl=3 (no response found!)
38	1.3386574…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=8/2048, ttl=3 (no response found!)
39	1.3386676…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=9/2304, ttl=3 (no response found!)
40	1.3386822…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=10/2560, ttl=4 (reply in 54)
41	1.3386945…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=11/2816, ttl=4 (reply in 55)
42	1.3387065…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=12/3072, ttl=4 (reply in 57)
43	1.3387235…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=13/3328, ttl=5 (reply in 59)
44	1.3387329…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=14/3584, ttl=5 (reply in 60)
45	1.3387450…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=15/3840, ttl=5 (reply in 61)
46	1.3387586…	172.26.50.102	193.136.9.240	ICMP	76	Echo (ping) request id=0x0001, seq=16/4096, ttl=6 (reply in 62)

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

É possível observar padrões nos campos *"Identification"* e *"Time to Live"*. O campo *"Identification"* lista o número de sequência do pacote e este apenas incrementa. Já o campo *"Time to Live"* incrementa a cada 3 pacotes, padrão responsável devido ao comportamento do comando *traceroute*, que envia 3 pacotes por cada TTL a ser testado.

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?

Ao observar os respetivos pacotes de resposta dos *routers*, verificamos que foram enviados 3 respostas por router (IP 172.16.115.252, 172.16.2.1 e 172.26.254.254). Ao analisar os pacotes descobrimos que o valor de TTL, apesar de ser igual nas 3 respostas repetidas, estas diferem entre si sendo, respetivamente, valores de 253, 254 e 255. Isto deve-se ao facto de, apesar do TTL ter expirado e o *router* enviar uma mensagem de aviso/erro, o valor de TTL é conservado, ou seja, conforme a mensagem retorna pela rede, o seu valor de TTL é decrementado por cada salto. Visto que, no cabeçalho do datagrama de IP, o valor de TTL é armazenado num *byte*, o seu valor varia entre 0 a 255, sendo que, decrementando um valor de 0 resultará num *underflow*, o que fará o valor retornar ao seu valor máximo possível de 255.

Figura 7: Captura de Pacotes ICMP TTL Exceeded

```
53 1.3737033… 172.16.115.252   172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
56 1.3737036… 172.16.115.252   172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
58 1.3737037… 172.16.115.252   172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
49 1.3736222… 172.16.2.1       172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
50 1.3736226… 172.16.2.1       172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
51 1.3736227… 172.16.2.1       172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
63 1.3737517… 172.26.254.254   172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
64 1.3737518… 172.26.254.254   172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
65 1.3737519… 172.26.254.254   172.26.50.102   ICMP     72 Time-to-live exceeded (Time to live exceeded in transit)
```

```
▸ Frame 53: 72 bytes on wire (576 bits), 72 bytes captured (576 bits)
▸ Linux cooked capture
▾ Internet Protocol Version 4, Src: 172.16.115.252, Dst: 172.26.50.102
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x914f (37199)
  ▸ Flags: 0x0000
    Fragment offset: 0
    Time to live: 253
    Protocol: ICMP (1)
    Header checksum: 0x2de8 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.115.252
    Destination: 172.26.50.102
▸ Internet Control Message Protocol
```

TTL Exceeded Router 172.16.115.252

```
▸ Frame 49: 72 bytes on wire (576 bits), 72 bytes captured (576 bits)
▸ Linux cooked capture
▾ Internet Protocol Version 4, Src: 172.16.2.1, Dst: 172.26.50.102
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0xc31b (49947)
  ▸ Flags: 0x0000
    Fragment offset: 0
    Time to live: 254
    Protocol: ICMP (1)
    Header checksum: 0x6d17 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.2.1
    Destination: 172.26.50.102
▸ Internet Control Message Protocol
```

TTL Exceeded Router 172.16.2.1

Figura 8: TTL Exceeded Router 172.16.254.254

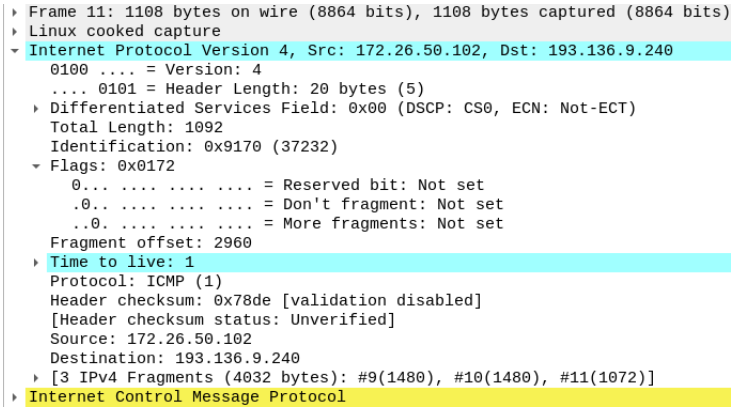
```
▸ Frame 64: 72 bytes on wire (576 bits), 72 bytes captured (576 bits)
▸ Linux cooked capture
▾ Internet Protocol Version 4, Src: 172.26.254.254, Dst: 172.26.50.102
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▸ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 56
    Identification: 0xbede (48862)
  ▸ Flags: 0x0000
    Fragment offset: 0
    Time to live: 255
    Protocol: ICMP (1)
    Header checksum: 0x728c [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.254.254
    Destination: 172.26.50.102
▸ Internet Control Message Protocol
```

1.3 Questão 3 - Análise fragmentação de pacotes Ipv4

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

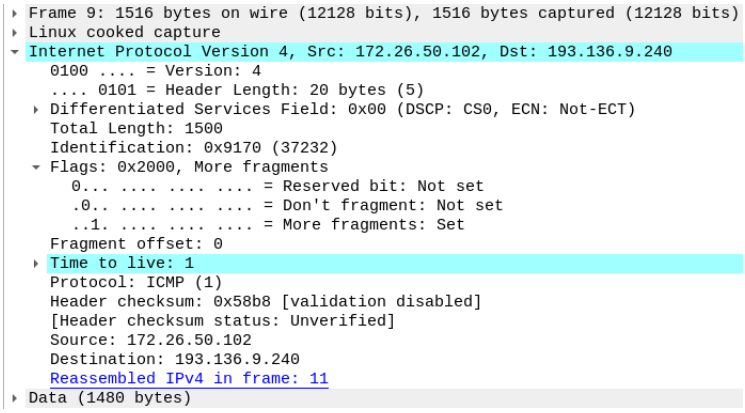
Selecionando o primeiro pacote ICMP capturado referente a situação ii), obtivemos os seguintes dados. Como podemos verificar através da análise do pacote, ocorreu a necessidade de fragmentar o datagrama em 3 segmentos, visto que o tamanho do datagrama enviado (4052 *bytes*) ultrapassa o tamanho das MTU's (1500 *bytes*).

Figura 9: Análise do Pacote ICMP com fragmentação (ii)



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?

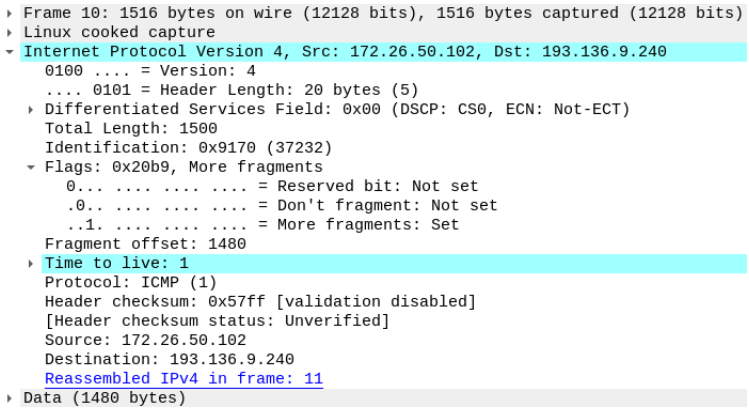
Figura 10: Análise do Pacote IPv4 (Primeiro fragmento)



Através das *flags* podemos observar que a *flag More fragments* foi definida como *Set*, indicando-nos que o datagrama foi fragmentado e que serão enviados mais fragmentos. Também podemos observar que este fragmento é o primeiro a ser enviado devido ao atributo *Fragment Offset* que toma um valor de 0, indicando que este é o primeiro segmento. Este datagrama tem um tamanho de 1500 *bytes* (verificado pelo campo *Total Length*), o que corresponde ao valor de MTU.

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?

Figura 11: Análise do Pacote IPv4 (Segundo fragmento)



Analisando o campo *Fragment Offset*, podemos concluir que este pacote não corresponde ao primeiro fragmento do datagrama enviado pois o seu valor é diferente de 0, ou seja, a informação contida neste pacote será reconstruida num *offset*, não sendo este pacote o primeiro fragmento no processo de reconstrução do datagrama original. No entanto, também concluimos que este pacote não será o último fragmento, visto que a *flag More Fragments* está definida como *Set*, indicando a receção de mais fragmentos

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

Foram criados ao todo 3 fragmentos a partir do datagrama original. Estes têm tamanhos respetivamente de 1500, 1500 e 1092 *bytes* cada.

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.

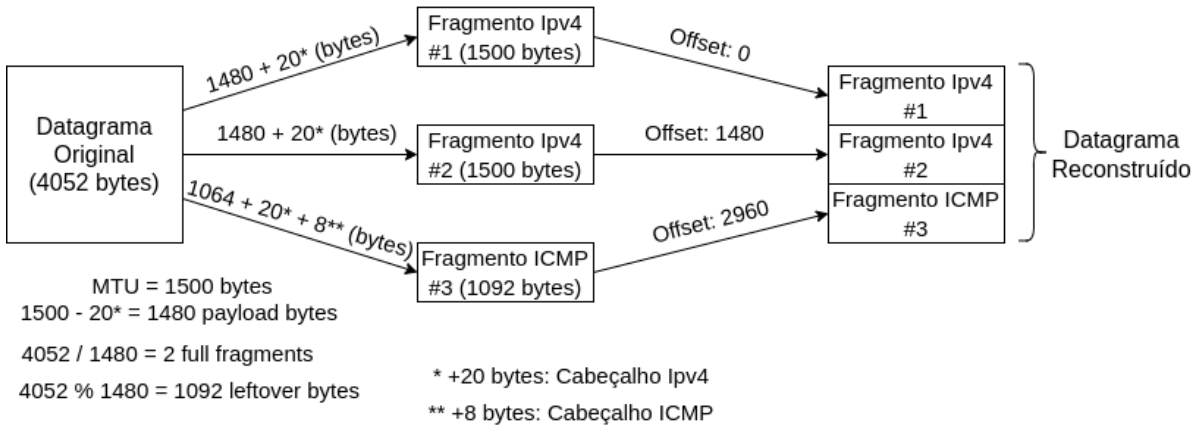
Entre os diferentes fragmentos, vários campos mudam no seus cabeçalhos IP, sendo os mais significativos o *Fragment Offset* e *Total Length*. O campo *Total length* é igual entre o primeiro e segundo pacote, sendo igual ao valor de MTU (1500 *bytes*), tendo o último pacote um tamanho de 1092 *bytes*. O campo *Fragment Offset* difere entre os 3 fragmentos, tendo os valores 0, 1480 e 2960 respetivamente.

Estes valores irão ajudar na reconstrução do datagrama original, visto que indicam o *local/offset* onde o *payload* de cada fragmento terá de ser "colocado"de forma a reconstruir a *payload* original. Podemos confirmar isto analisando o *offset* entre os fragmentos, pois estes contêm valores múltiplos de 1480, ou seja, o valor máximo de um pacote IPv4 menos o seu cabeçalho (1500 - 20 = 1480), obtendo assim o tamanho da *payload* de cada fragmento.

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

Para este exercício, decidimos realizar um esquema para demonstrar o processo de fragmentação:

Figura 12: Cálculo de fragmentação



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

Para detetar o último fragmento do datagrama é necessário capturar o fragmento que contenha o campo *"fragment offset"* diferente de 0, o que indica que o datagrama foi fragmentado, e a *flag "More fragment"* definida como *"Not Set"*, indicando que este é o último fragmento enviado. Também é necessário verificar que o campo *Identification* do último fragmento é igual ao do datagrama original.

Uma forma de detetar estes fragmentos a partir do programa "WireShark", após uma captura, será utilizando o fitro *"ip.flags.mf == "not set"and ip.frag_offset != 0"*.

1.4 Questão 1 - Estabelecimento da Topologia Core

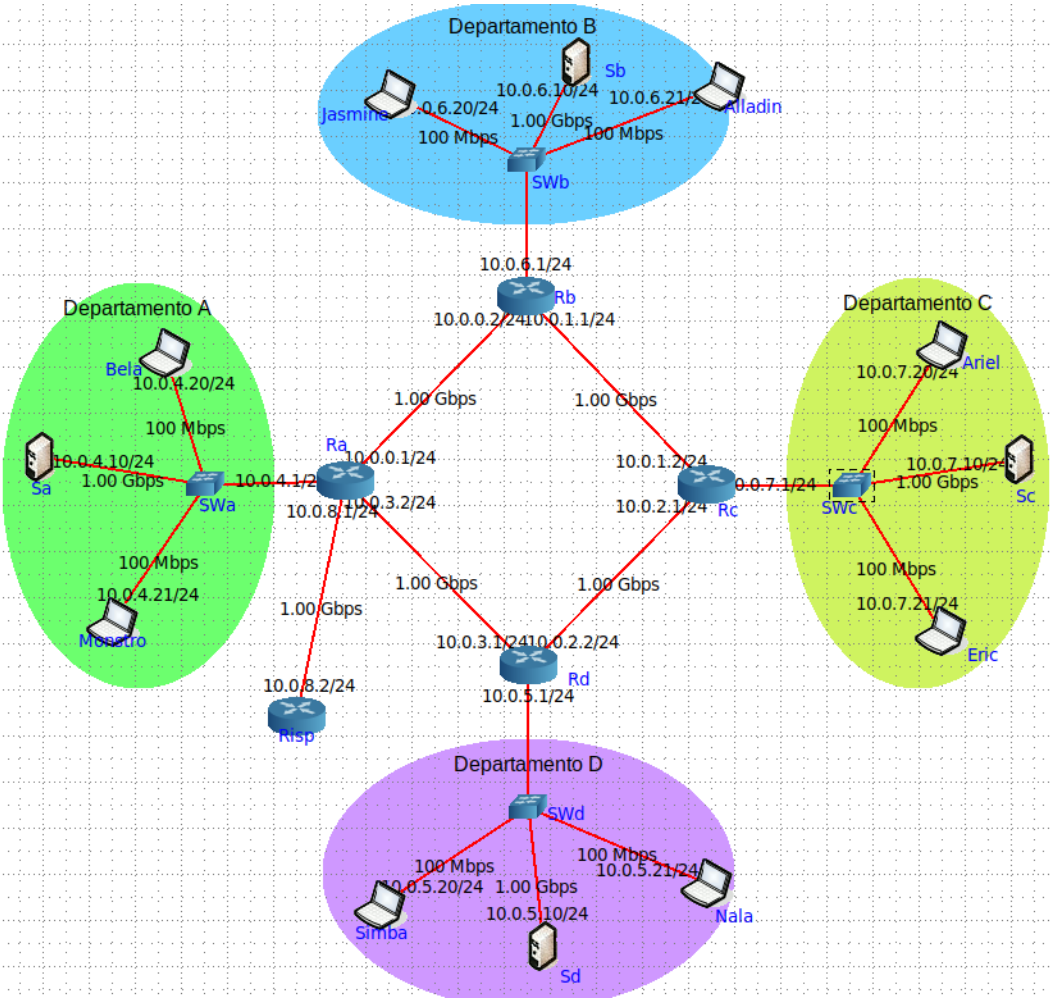
a. Indique que endereços IP e máscaras de rede foram atribuídos pelo CORE a cada equipamento. Para simplificar, pode incluir uma imagem que ilustre de forma clara a topologia definida e o endereçamento usado.

Depois de construir a topologia como especificado, verificamos que todos os IP's atribuidos utilizam a mesma máscara **/24** (255.255.255.0). Também reparamos que, as rede dentro dos departamentos partilham um mesmo prefixo de IP, ou seja, fazem parte da mesma sub-rede.

- **Departamento A:** 10.0.4.0
- **Departamento B:** 10.0.6.0
- **Departamento C:** 10.0.7.0
- **Departamento D:** 10.0.5.0

Apresentamos de seguida uma imagem com a topologia desenvolvida:

Figura 13: Topologia Core



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

Estes endereços são privados, visto que os endereços atribuídos pelo CORE fazem parte de um bloco de IP's reservados (IP 10.0.0.0/8). Este bloco de IP's, assim como outros, são reservados e atribuídos pela IANA (Internet Assigned Numbers Authority), sende este bloco destinado para o endereçamento de IP's privados a serem utilizados dentro de uma rede interna.

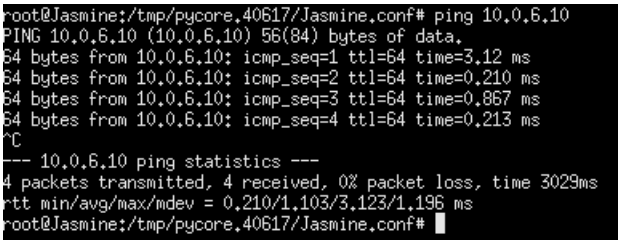
c. Porque razão não é atribuído um endereço IP aos switches?

Estes não contêm endereços IP's visto que são componentes de nível inferior na topologia de rede (*Link*). Estes *switches* apenas auxiliam no envio dos pacotes aos recipientes/equipamentos corretos através dos endereços MAC, não sendo necessário atribuir um endereço de rede.

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

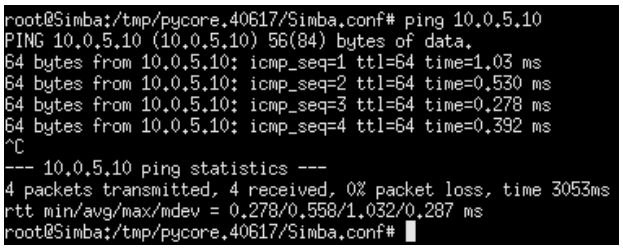
Para este exercício, utilizamos o comando *ping* para testar a conectividade interna de cada departamento (sub-rede), sendo assim necessário executar este comando em cada departamento. Como os resultados seguintes demonstram, todos os departamentos conseguem comunicar com o seu respetivo servidor:

Ping Bela → *ServidorA*



Ping Jasmine → *ServidorB*

Ping Ariel → *ServidorC*



Ping Simba → *ServidorD*

e. Execute o número mínimo de comandos ping que lhe permite verificar a existência de conetividade IP entre departamentos.

Para este exercício recorremos a 3 comandos *pings* para verificar a conectividade entre todos os departamentos. Começamos por testar a conectividade do Departamento D com o Departamento B, de seguida a conectividade do Departamento A com o Departamento C e, por último, e de forma a interligar de forma mínima os vários departamentos, a conectividade do Departamento B com o Departamento A.

Apresentamos de seguida os resultados obtidos, onde todas as conexões foram estabelecidas com sucesso:

```
root@Simba:/tmp/pycore.40617/Simba.conf# ping 10.0.6.10
PING 10.0.6.10 (10.0.6.10) 56(84) bytes of data.
64 bytes from 10.0.6.10: icmp_seq=1 ttl=61 time=2.06 ms
64 bytes from 10.0.6.10: icmp_seq=2 ttl=61 time=0.512 ms
64 bytes from 10.0.6.10: icmp_seq=3 ttl=61 time=0.544 ms
64 bytes from 10.0.6.10: icmp_seq=4 ttl=61 time=0.701 ms
^C
--- 10.0.6.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3037ms
rtt min/avg/max/mdev = 0.512/0.953/2.058/0.641 ms
root@Simba:/tmp/pycore.40617/Simba.conf#
```

Ping Simba → *ServidorB*

```
root@Bela:/tmp/pycore.40617/Bela.conf# ping 10.0.7.10
PING 10.0.7.10 (10.0.7.10) 56(84) bytes of data.
64 bytes from 10.0.7.10: icmp_seq=1 ttl=61 time=1.64 ms
64 bytes from 10.0.7.10: icmp_seq=2 ttl=61 time=0.508 ms
64 bytes from 10.0.7.10: icmp_seq=3 ttl=61 time=0.481 ms
^C
--- 10.0.7.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2024ms
rtt min/avg/max/mdev = 0.481/0.875/1.638/0.539 ms
root@Bela:/tmp/pycore.40617/Bela.conf#
```

Ping Bela → *ServidorC*

```
root@Jasmine:/tmp/pycore.40617/Jasmine.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
64 bytes from 10.0.4.10: icmp_seq=1 ttl=62 time=0.864 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=62 time=0.355 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=62 time=0.396 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=62 time=0.361 ms
^C
--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3041ms
rtt min/avg/max/mdev = 0.355/0.494/0.864/0.214 ms
root@Jasmine:/tmp/pycore.40617/Jasmine.conf#
```

Ping Jasmine → *ServidorA*

f. Verifique se existe conectividade IP do portátil Bela para o router de acesso R_ISP.

Na resolução deste exercício, executamos um simples comando *ping* para o router de acesso, obtendo sucesso nesta comunicação:

Figura 14: Ping A → *R_isp*

```
root@Bela:/tmp/pycore.40617/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=1.26 ms
64 bytes from 10.0.8.2: icmp_seq=2 ttl=63 time=0.396 ms
64 bytes from 10.0.8.2: icmp_seq=3 ttl=63 time=0.302 ms
64 bytes from 10.0.8.2: icmp_seq=4 ttl=63 time=0.309 ms
64 bytes from 10.0.8.2: icmp_seq=5 ttl=63 time=0.290 ms
^C
--- 10.0.8.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4081ms
rtt min/avg/max/mdev = 0.290/0.510/1.256/0.374 ms
root@Bela:/tmp/pycore.40617/Bela.conf#
```

1.5 Questão 2 - Para o router R_A 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 respetivo (man netstat).

Figura 15: Tabela de encaminhamento do Router Ra

```
root@Ra:/tmp/pycore.40617/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        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.2.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth1
10.0.3.0        0.0.0.0         255.255.255.0   U         0 0          0 eth1
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth2
10.0.5.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth1
10.0.6.0        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.7.0        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.8.0        0.0.0.0         255.255.255.0   U         0 0          0 eth3
root@Ra:/tmp/pycore.40617/Ra.conf#
```

Figura 16: Tabela de encaminhamento do Host Bela

```
root@Bela:/tmp/pycore.42401/Bela.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@Bela:/tmp/pycore.42401/Bela.conf#
```

Como verificamos, as tabelas de encaminhamento contêm 8 campos:

- **Destination -** Endereço IP de destino
- **Gateway -** Endereço IP do próximo salto
- **Genmask -** Máscara do endereço de IP de destino
- **Flags -** Flags de informação. EX's: **U -** Rota disponível; **G -** Utilização da Gateway
- **MSS -** (Maximum Segment Size): Tamanho máximo de segmentos TCP nesta rota
- **Window -** Tamanho por defeito da janela de conexões TCP nesta rota
- **IRTT -** Initial RTT (Round Trip Time)
- **IFace -** Interface de acesso a ser utilizada nesta rota

Nas entradas das tabelas, podemos verificar que, aquelas que possuem uma *gateway* com valor 0.0.0.0 correspondem a sub-redes/destinos os quais o Router A faz parte, sendo esta *gateway* "default"indicativa disso. Por sua vez, as restantes interfaces diferem, correspondendo as respetivas "saídas"do *router* para essa sub-rede.

Nas restantes entradas, observamos que, cada destino de sub-rede contém uma *gateway* específica que servirá como o "próximo salto"para alcançar o destino. Também podemos observar a *flag* G nestas entradas, dando indicação que estas *gateway* terão de ser utilizadas.

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)

Num modelo de tabelas de encaminhamento estáticas, estas não se alteram à exceção do pedido/comando do admnistrador responsável. Por sua vez, num modelo de tabelas de encaminhamento dinâmico, como esta está em constante atualização com os restantes *routers*, é necessário algum processo que faça este controlo e atualização. Sabendo isto, pretendemos procurar provas do modelo de encaminhamento utilizado pelas máquinas "Bela" e "Ra" analisando os processos que estão a correr em cada sistema.

Analisando os processos a decorrer em ambas as máquinas ("Bela" e "Ra"), através do comando "ps -ax", obtemos os seguintes resultados:

Figura 17: "ps -ax"Host Bela

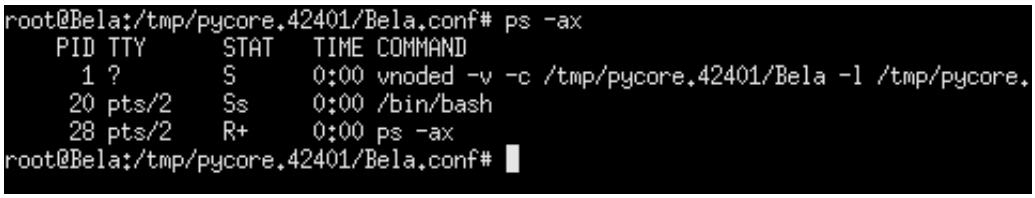
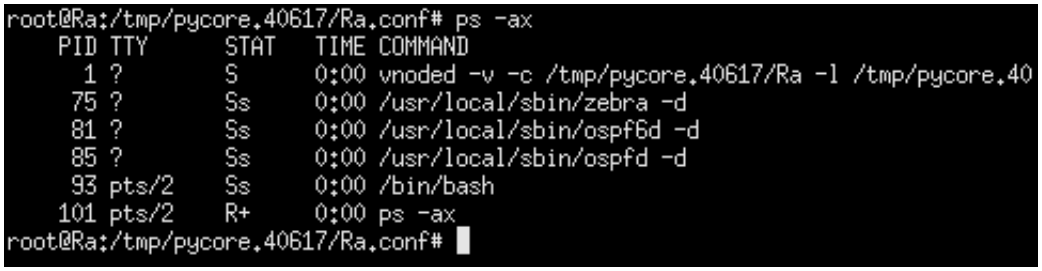


Figura 18: "ps -ax"Router A

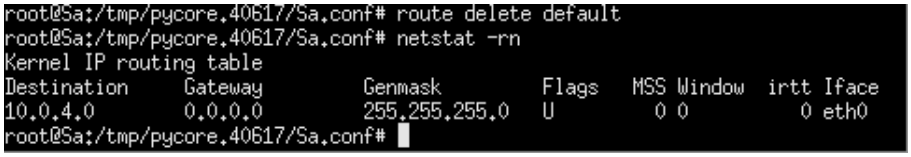


Com estes resultados, conseguimos concluir que o *Host* "Bela" possui um modelo de encaminhamento estático visto que não possui nenhum processo de controlo de rotas. Este resultado é esperado visto que, normalmente, os *hosts* são os *"endpoints"* de uma rede, sendo este *end point* fixo, não havendo necessidade de processos de atualização dinâmica.

Já para o "Router A", podemos verificar a existência de vários processos responsáveis pelo controlo e atualização da tabela de encaminhamento (processos OSPF - **Open Shortest Path First**), provando que esta tem um funcionamento dinâmico. Este resultado também é esperado visto que numa rede, há constante modificações na topologia sendo necessário que estas mudanças sejam atualizadas entre os vários *routers* na rede.

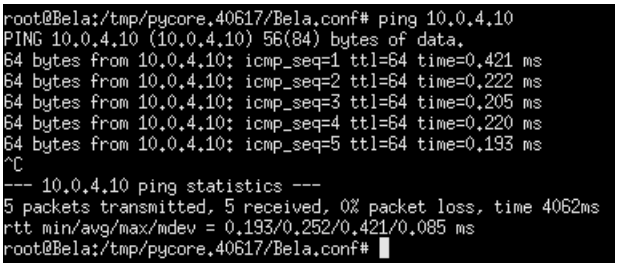
c. Admita que, por questões administrativas, a rota por defeito (0.0.0.0 ou default) deve ser retirada definitivamente da tabela de encaminhamento do servidor SA. Use o comando route delete para o efeito. Que implicações tem esta medida para os utilizadores da LEI-RC que acedem ao servidor. Justifique.

Figura 19: Remoção da rota por defeito do Servidor A



Depois de removido a rota por defeito do Servidor "Sa", verificamos que, para equipamentos dentro do mesmo departamento (neste caso A), não ocorreu nenhuma mudança significativa, sendo o *host* "Bela" capaz de comunicar com o Servidor "A".

Figura 20: Ping Bela $\rightarrow$ *ServidorA*



Esta comunicação ocorre com sucesso pois na tabela de encaminhamento do Servidor "A", a sua única entrada corresponde ao destino de IP 10.0.4.0, sendo este IP pertencente à sub-rede do departamento A, logo, o Servidor "A" é capaz de comunicar com máquinas que pertençam a esta sub-rede.

De seguida, exprimentamos analisar a conetividade do Servidor "A"com equipamentos fora do seu departamento:

Figura 21: Ping Nala $\rightarrow$ *ServidorA*

Como podemos verificar, não foi possível estabelecer a conexão ao Servidor "A". Para além do mais, podemos observar que, apesar dos pacotes terem sido enviados com sucesso, nenhum destes foram enviados de volta. Isto deve-se à falta de entradas da tabela de encaminhamento no Servidor "A" (neste caso, a rota por defeito), sendo este incapaz de mandar tráfego para a sub-rede 10.0.4.0 (e para as restantes, à exceção da sua sub-rede).

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.

Para este exercício, começamos por analisar a causa das falhas de comunicação observadas no exercício anterior e pudemos concluir que, com a remoção da rota por defeito, o Servidor "A" perdeu a capacidade de comunicar para o exterior da sua sub-rede. Para resolver esta falha, temos como objetivo acrescentar várias entradas na tabela de encaminhamento do Servidor "A", de forma a que esta conheça todas as sub-redes na topologia.

Inicialmente, começamos por adicionar algumas sub-redes através do comando (por exemplo) *route add 10.0.0.0 gw 10.0.4.1 eth0*, no entanto ao analisar a tabela de encaminhamento, encontramos o seguinte:

Figura 22: Tabela de encaminhamento do Servidor A

```
root@Sa:/tmp/pycore.42967/Sa.conf# route add 10.0.0.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.4.1        255.255.255.255 UGH       0 0          0 eth0
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
root@Sa:/tmp/pycore.42967/Sa.conf#
```

Observamos que a entrada criada está incorreta, pois não só contém uma máscara incorreta, como também observamos a presença da *flag* **H**, indicativo que o endereço de destino é um *host*.

Para remediar o problema, eliminamos a entrada recém-criada e, após alguma investigação do comando "route add", descobrimos como definir uma máscara específica, assim como indicar que o endereço de IP pertence a uma rede. (Comando completo: *route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.4.1 eth0*).

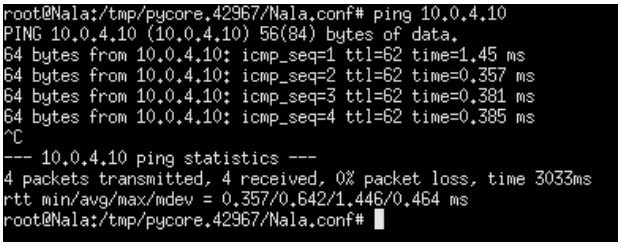
Figura 23: Adição de rotas no Servidor A

```
root@Sa:/tmp/pycore.42967/Sa.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@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.4.1        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 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.4.1 eth0
root@Sa:/tmp/pycore.42967/Sa.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.1.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.2.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.3.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
10.0.4.0        10.0.4.1        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 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@Sa:/tmp/pycore.42967/Sa.conf#
```

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.

Para testar a conetividade do Servidor "A", executamos um comando *ping* através de um *host* localizado no próprio departamento do servidor (*Host* "Bela"), um outro *ping* através de um *host* localizado fora da sub-rede do servidor (*Host* "Nala"), e por último um *ping* a partir do próprio servidor, obtendo sucesso em todas as ocasiões. Apresentamos também a tabela de encaminhamento final.

Ping Host Bela → *ServidorA*



Ping Host Nala → *ServidorA*

Figura 24: Ping Servidor A → *Alladin*

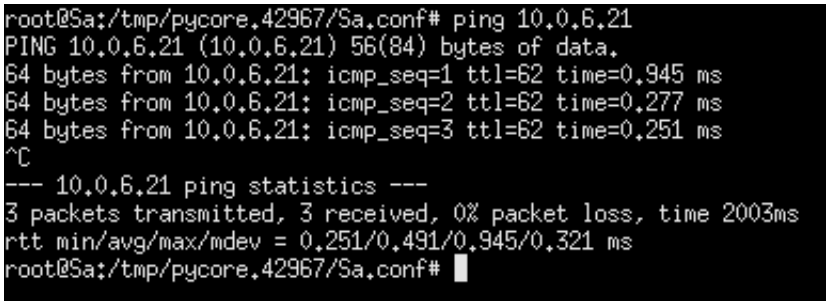


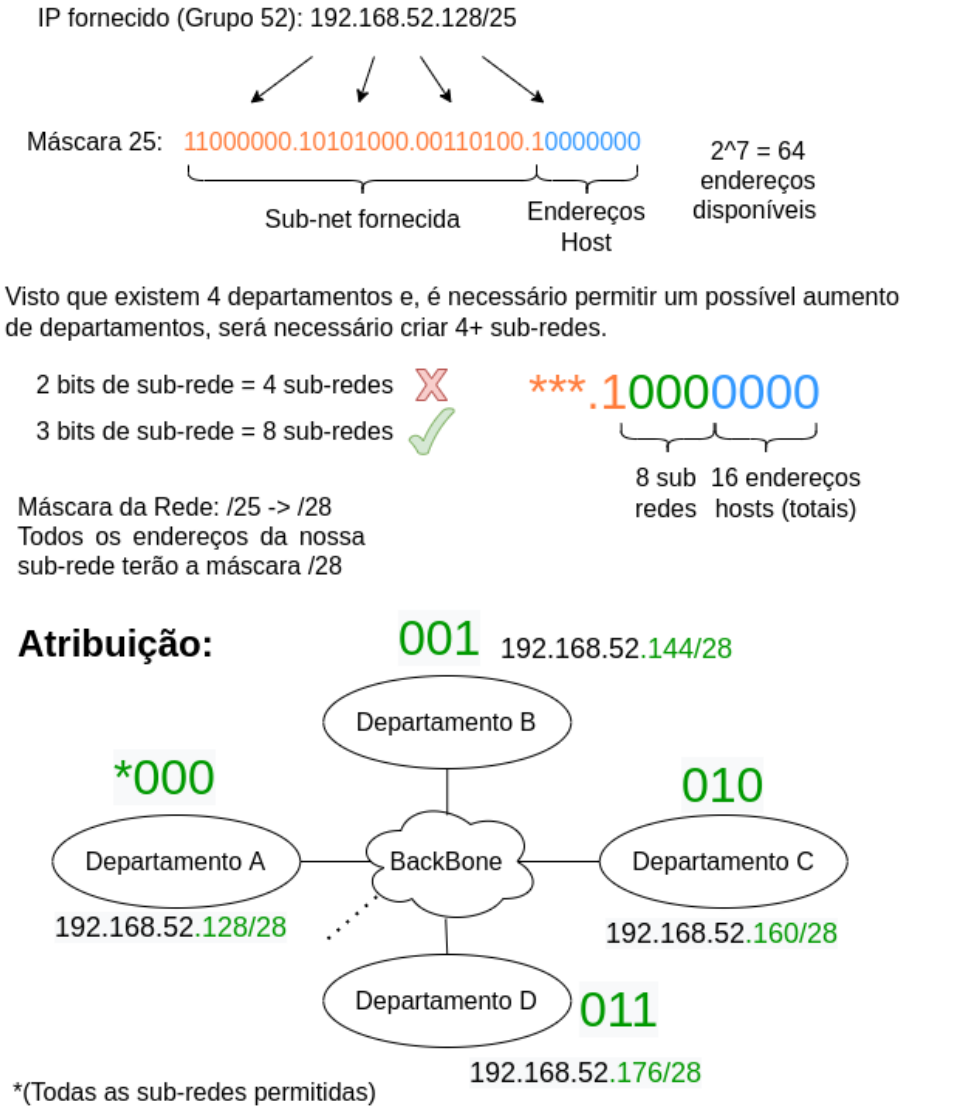
Figura 25: Tabela final de encaminhamento do Servidor A

1.6 Questão 3 - Definição de Sub-redes

1. **Considere que dispõe apenas do endereço de rede IP 192.168.XXX.128/25, em que XXX é o decimal correspondendo ao seu número de grupo (PL52). 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 subredes são usáveis. Justifique as opções tomadas no planeamento.**

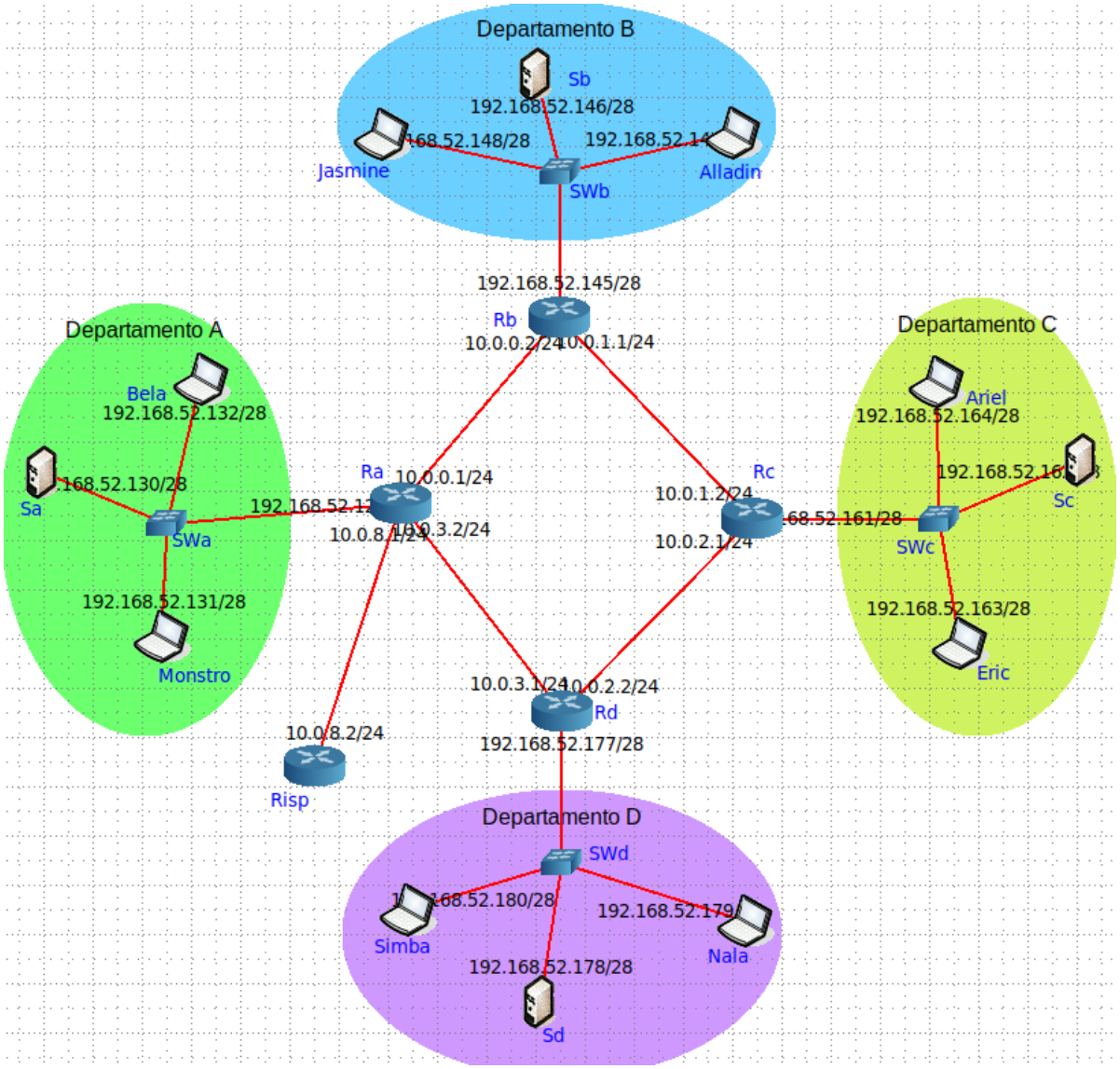
Apresentamos um esquema com as decisões tomadas face este problema:

Figura 26: Subnetting 192.168.52.128/25



Tendo em conta este planeamento, alteramos os vários endereços IP's dos equipamentos existentes nos departamentos, sendo esta a topologia final obtida:

Figura 27: Topologia Subnetted



- **Router A:** 192.168.52.129
- **Servidor A:** 192.168.52.130
- **Monstro:** 192.168.52.131
- **Bela:** 192.168.52.132

- **Router B:** 192.168.52.145
- **Servidor B:** 192.168.52.146
- **Alladin:** 192.168.52.147
- **Jasmine:** 192.168.52.148

- **Router C:** 192.168.52.161
- **Servidor C:** 192.168.52.162
- **Eric:** 192.168.52.163
- **Ariel:** 192.168.52.164

- **Router D:** 192.168.52.177
- **Servidor D:** 192.168.52.178
- **Nala:** 192.168.52.179
- **Simba:** 192.168.52.180

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

Visto que foi necessário utilizar 3 *bits* para realizar o *subnetting* da nossa rede, a máscara final, tendo em conta a máscara já utilizada pelo ISP **/25**, tem um valor de **/28** (255.255.255.240).

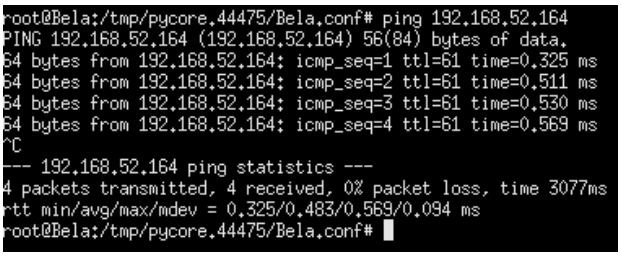
Cada departamento contém 4 *bits* para endereçamento de IP's, dando um total de 16 endereços totais, no entanto, 2 destes endereços (0000 e 1111) são reservados à sub-rede, estando apenas disponíveis 14 endereços IP para os *hosts* por departamento.

Com a utilização de 3 *bits* para prefixos de sub-redes, podem ser criadas no total 8 sub-redes, assumindo que as sub-redes 192.168.52.128/28 e 192.168.52.240/28 (000 e 111) podem ser utilizadas. Como neste exemplo foram utilizadas 4 sub-redes (uma por cada departamento), ficam disponíveis mais 4 sub-redes para expansão futura.
(**Sub-redes:** 192.168.52.192/28 ; 192.168.52.208/28 ; 192.168.52.224/28 ; 192.168.52.240/28)

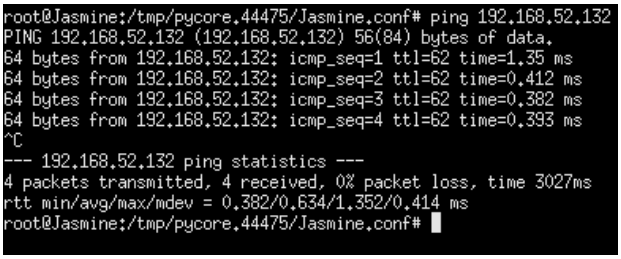
3. Verifique e garanta que a conectividade IP interna na rede local LEI-RC é mantida. No caso de não existência de conetividade, reveja a atribuição de endereços efetuada e eventuais erros de encaminhamento por forma a realizar as correções necessárias. Explique como procedeu.

Para efetuar este teste de conetividade, efetuamos vários *pings*, não só entre equipamentos da mesma sub-rede, como também equipamentos entre várias sub-redes, semelhante à resolução da questão 1.E

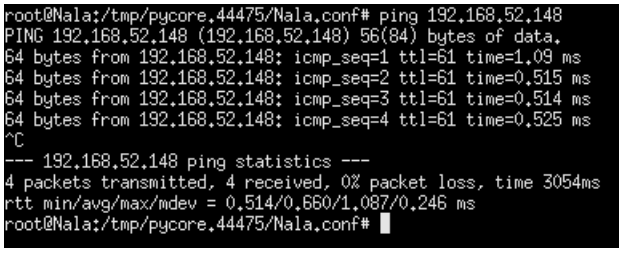
Ping Host Bela → *ServidorA*



Ping Host Bela → *HostAriel*



Ping Host Jasmine → *Bela*



Ping Host Nala → *HostJasmine*

2 Conclusão

Com o terminar desta trabalho prático, encontramo-nos satisfeitos com o trabalho desenvolvido, tendo alcançado todas as metas propostas pelos docentes, assim como ter justificado adequadamente os resultados observados e analisados. No entanto, existem possíveis melhoramentos a serem feitos, como uma estruturação mais cuidada de algumas respostas, assim como a utilização mais correta dos termos da área de redes.

Este trabalho também nos permitiu aprofundar o nosso conhecimento na área de redes através da aplicação prática dos conteúdos lecionados nas aulas téoricas. Desenvolvemos uma melhor perceção acerca do funcionamento dos sistemas de endereçamentos e *routing* das redes, sendo estes pilares fundamentais para o bom funcionamento não só de redes privadas como a *internet* em geral. Também pudemos observar e compreender de forma mais clara o funcionamento do protocolo *IPv4*, graças às várias análises efetuadas ao longo do trabalho.