Universidade do Minho

Engenharia Informática

21/22

Redes - TP2

Afonso Amorim A97569
Luís Ferreira A95111
Pedro Dantas A97396

Indice

1 Parte 1

1.1 Exercício 1

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.

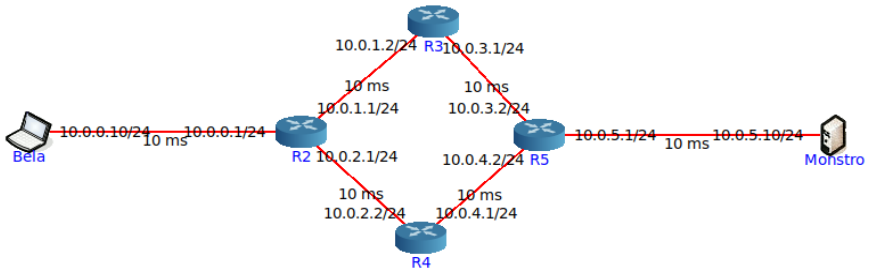


Fig. 1 - Modelo topologia CORE

1.1.1 Alínea a)

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

```
root@Bela:/tmp/pycore.34439/Bela.conf# traceroute -I 10.0.5.10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  40.431 ms  40.345 ms  40.340 ms
 2  10.0.1.2 (10.0.1.2)  82.120 ms  82.120 ms  82.119 ms
 3  10.0.3.2 (10.0.3.2)  102.891 ms  102.889 ms  102.888 ms
 4  10.0.5.10 (10.0.5.10)  143.692 ms  143.691 ms  143.689 ms
root@Bela:/tmp/pycore.34439/Bela.conf#
```

Fig. 2 - Execução do comando traceroute

1.1.2 Alínea 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.

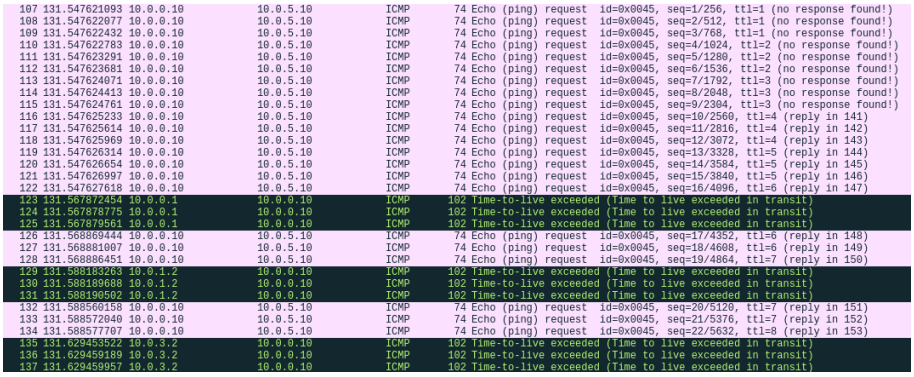


Fig. 3 - Pacotes Recebidos

Como podemos ver, os pacotes são enviados em conjuntos de 3. Podemos também observar que os primeiros 3 conjuntos (TTL < 4) não conseguem chegar ao destino, pelo que devolvem uma mensagem de erro.

1.1.3 Alínea 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.

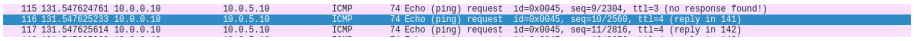


Fig. 4 - TTL Mínimo

Como pudemos verificar no exercício anterior, é necessário um TTL mínimo igual a 4.

1.1.4 Alínea 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.

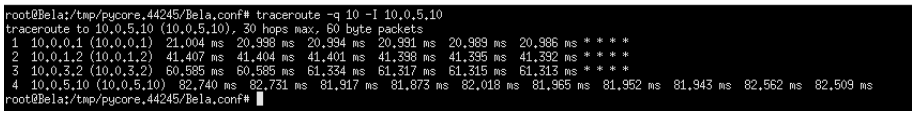


Fig. 5 - Tempo de ida-e-volta

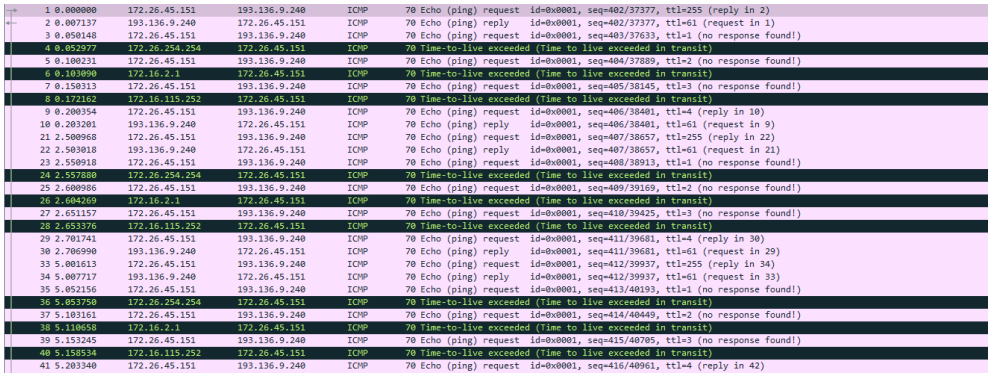
Como podemos verificar na Figura 5, enviamos 10 pacotes e recebemos o tempo que demoram a ir e voltar. Para calcular o tempo médiio de ida-e-volta utilizamos os valores na *linha 4* e fazemos a média de todos eles.
(82.740+82.731+81.917+81.873+81.917+81.873+82.018+81.965+81.952+81.943+82.562+82.509)/10
Ficamos assim a saber que o tempo médio de ida-e-volta é de 82.221 ms.

1.1.5 Alínea 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, o valor não pode ser calculado dividindo o RTT por dois porque ao fazer isso estaríamos a assumir que o tempo de ida e volta eram exatamente iguais, o que pode não acontecer. Ao voltar o pacote pode tomar um caminho diferente do caminho de ida e demorar mais ou menos tempo.

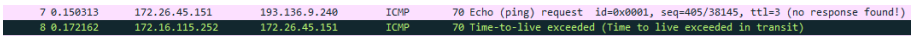
1.2 Exercício 2



1	0.000000	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=402/37377, ttl=255 (reply in 2)
2	0.007137	193.136.9.240	172.26.45.151	ICMP	70 Echo (ping) reply id=0x0001, seq=402/37377, ttl=61 (request in 1)
3	0.050148	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=403/37633, ttl=1 (no response found!)
4	0.052977	172.26.254.254	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
5	0.100231	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=404/37889, ttl=2 (no response found!)
6	0.103090	172.16.2.1	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
7	0.150313	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=405/38145, ttl=3 (no response found!)
8	0.172162	172.16.115.252	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
9	0.200354	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=406/38401, ttl=4 (reply in 10)
10	0.203201	193.136.9.240	172.26.45.151	ICMP	70 Echo (ping) reply id=0x0001, seq=406/38401, ttl=61 (request in 9)
21	2.500968	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=407/38657, ttl=255 (reply in 22)
22	2.503018	193.136.9.240	172.26.45.151	ICMP	70 Echo (ping) reply id=0x0001, seq=407/38657, ttl=61 (request in 21)
23	2.550918	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=408/38913, ttl=1 (no response found!)
24	2.557880	172.26.254.254	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
25	2.600986	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=409/39169, ttl=2 (no response found!)
26	2.604269	172.16.2.1	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
27	2.651157	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=410/39425, ttl=3 (no response found!)
28	2.653376	172.16.115.252	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
29	2.701741	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=411/39681, ttl=4 (reply in 30)
30	2.706990	193.136.9.240	172.26.45.151	ICMP	70 Echo (ping) reply id=0x0001, seq=411/39681, ttl=61 (request in 29)
33	5.001613	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=412/39937, ttl=255 (reply in 34)
34	5.007717	193.136.9.240	172.26.45.151	ICMP	70 Echo (ping) reply id=0x0001, seq=412/39937, ttl=61 (request in 33)
35	5.052156	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=413/40193, ttl=1 (no response found!)
36	5.053750	172.26.254.254	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
37	5.103161	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=414/40449, ttl=2 (no response found!)
38	5.110658	172.16.2.1	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
39	5.153245	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=415/40705, ttl=3 (no response found!)
40	5.158534	172.16.115.252	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)
41	5.203340	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=416/40961, ttl=4 (reply in 42)

Fig. 6 - Tráfego ICMP

1.2.1 Alínea a)



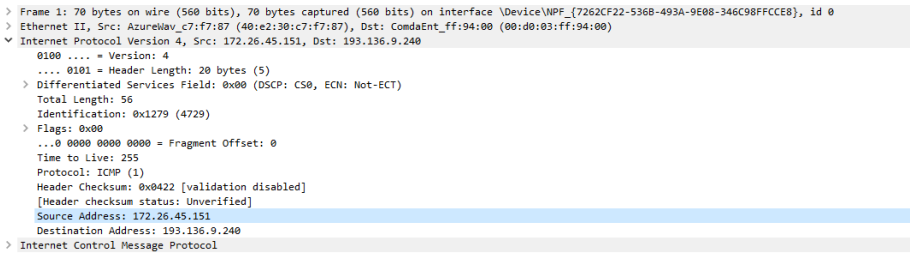
7	0.150313	172.26.45.151	193.136.9.240	ICMP	70 Echo (ping) request id=0x0001, seq=405/38145, ttl=3 (no response found!)
8	0.172162	172.16.115.252	172.26.45.151	ICMP	70 Time-to-live exceeded (Time to live exceeded in transit)

Fig. 7 - IP

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

Como é possível observar na figura 7, o endereço IP da interface ativa do nosso computador é: **172.26.45.151**

1.2.2 Alínea b)



```
> Frame 1: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface \Device\NPF_{7262CF22-536B-493A-9E08-346C98FFCCE8}, id 0
> Ethernet II, Src: AzureWav_c7:f7:87 (40:e2:30:c7:f7:87), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
v Internet Protocol Version 4, Src: 172.26.45.151, Dst: 193.136.9.240
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x1279 (4729)
  > Flags: 0x00
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0x0422 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.45.151
    Destination Address: 193.136.9.240
> Internet Control Message Protocol
```

Fig. 8 - Valores dos campos de um Pacote

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

Como podemos verificar na figura 8, o valor do campo protocolo é **1**, sendo que este identifica o protocolo **ICMP** (*Internet Control Message Protocol*).

1.2.3 Alínea c)

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

Através da análise da figura 8, podemos conferir que o cabeçalho IP(v4) possui um tamanho de 20 *bytes*. Tendo em conta que o campo do datagrama pode ser obtido através da diferença entre o tamanho do pacote (56 *bytes*) e o tamanho do cabeçalho, concluímos que este tem um tamanho de 36 *bytes*.

1.2.4 Alínea d)

O datagrama IP foi fragmentado? Justifique.

Tendo em conta novamente a figura 8, podemos verificar que o pacote não foi fragmentado. Isto porque tanto o valor da *flag* como o do *fragment offset* estão a 0.

1.2.5 Alínea e)

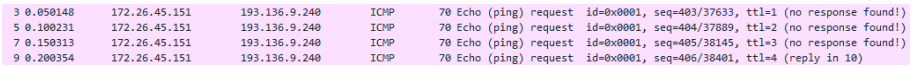


Fig.9 - Série de Pacotes Ordenados pelo Endereço Fonte

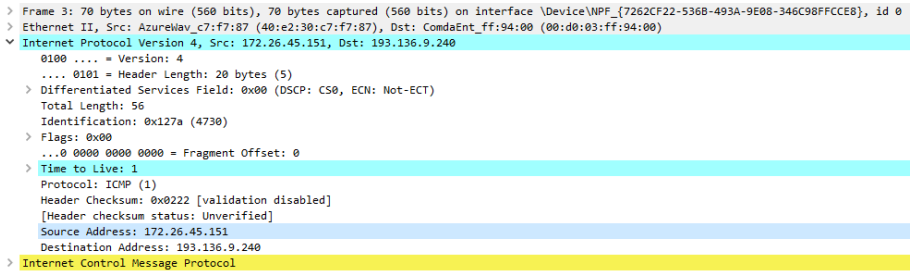


Fig. 10 - Pacote 1

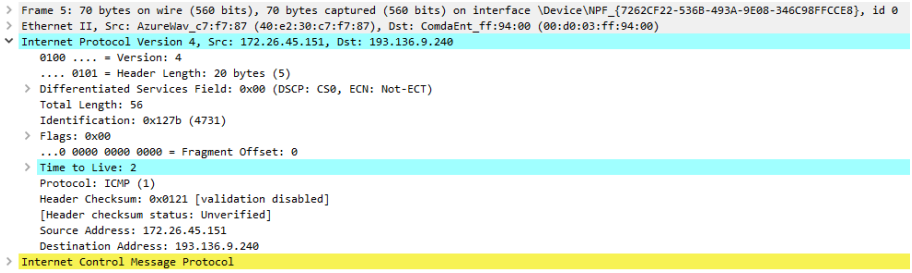


Fig. 11 - Pacote 2

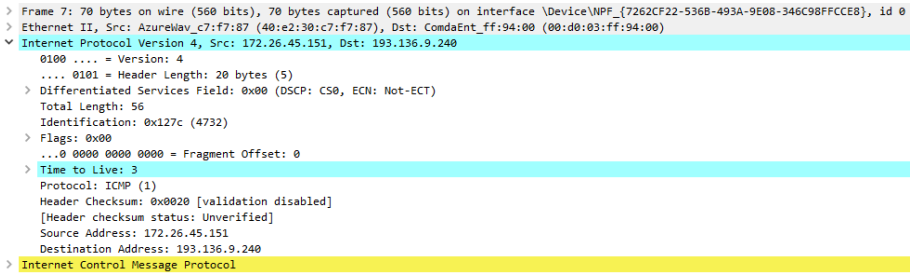


Fig. 12 - Pacote 3

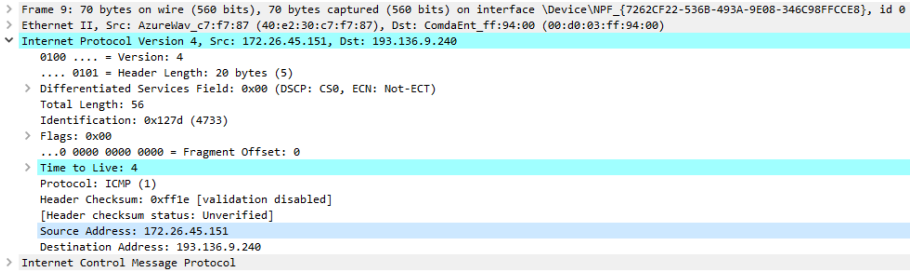


Fig. 13 - Pacote 4

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.

Como podemos observar nas figuras 10, 11, 12 e 13, os campos do cabeçalho IP que variam de pacote para pacote são o ***TTL*** e o ***Identification***.

1.2.6 Alínea f)

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

Analisando as figuras anteriormente referidas podemos verificar que, de facto, existe um padrão nos campos referidos, sendo que ambos têm um incremento de **1** de pacote em pacote.

1.2.7 Alínea g)

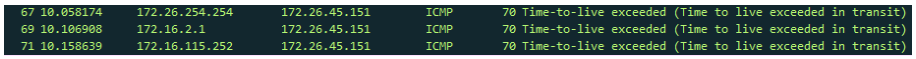


Fig.14 - Série de Pacotes Ordenados Pelo Endereço Destino

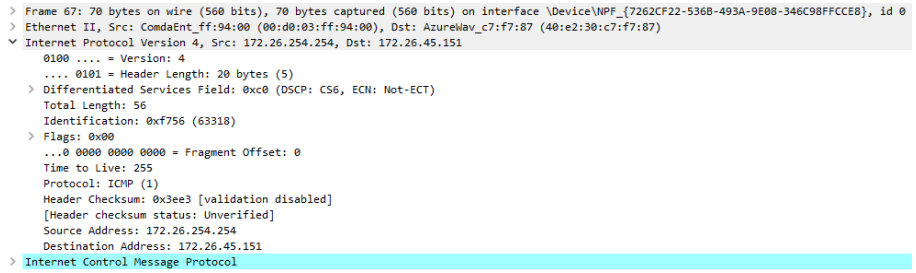


Fig. 15 - Pacote 1

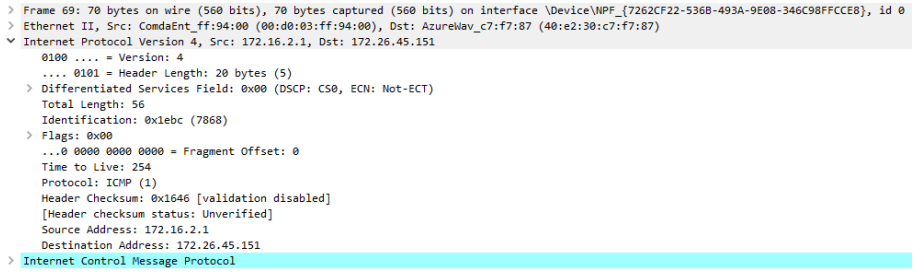


Fig.16 - Pacote 2

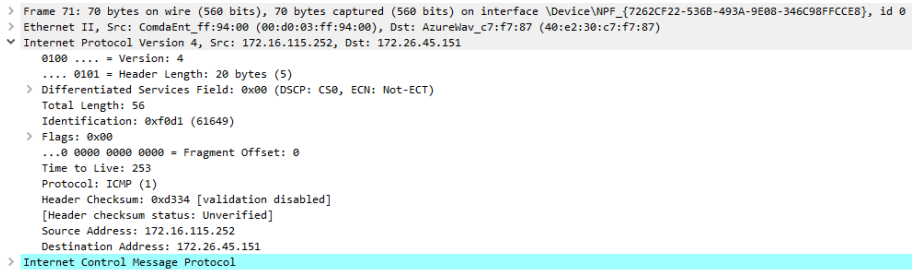
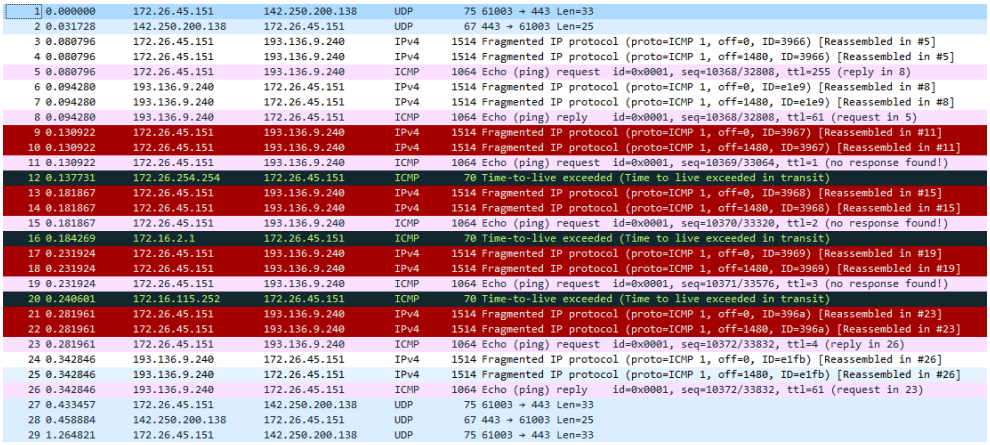


Fig. 17 - *Pacote 3*

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?

As figuras 15, 16 e 17 mostram, entre outros campos, os valores ***TTL*** dos respectivos pacotes. Assim, podemos observar que o primeiro pacote tem $\boldsymbol{TTL = 255}$, o segundo $\boldsymbol{TTL = 254}$ e o terceiro $\boldsymbol{TTL = 253}$. A diminuição neste campo deve-se ao facto de que, quando é necessário o envio de uma mensagem de erro, os pacotes são enviados de *routers* mais distantes, fazendo com que, no caminho de regresso, estes passem por mais *routers* intermédios, contribuindo então para o decremento referido.

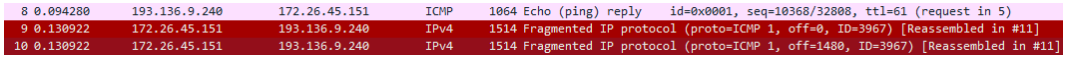


1	0.000000	172.26.45.151	142.250.200.138	UDP	75	61003 → 443 Len=33
2	0.031728	142.250.200.138	172.26.45.151	UDP	67	443 → 61003 Len=25
3	0.080796	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=3966) [Reassembled in #5]
4	0.080796	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3966) [Reassembled in #5]
5	0.080796	172.26.45.151	193.136.9.240	ICMP	1064	Echo (ping) request id=0x0001, seq=10368/32808, ttl=255 (reply in 8)
6	0.094280	193.136.9.240	172.26.45.151	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=e1e9) [Reassembled in #8]
7	0.094280	193.136.9.240	172.26.45.151	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e1e9) [Reassembled in #8]
8	0.094280	193.136.9.240	172.26.45.151	ICMP	1064	Echo (ping) reply id=0x0001, seq=10368/32808, ttl=61 (request in 5)
9	0.130922	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=3967) [Reassembled in #11]
10	0.130922	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3967) [Reassembled in #11]
11	0.130922	172.26.45.151	193.136.9.240	ICMP	1064	Echo (ping) request id=0x0001, seq=10369/33064, ttl=1 (no response found!)
12	0.137731	172.26.254.254	172.26.45.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
13	0.181867	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=3968) [Reassembled in #15]
14	0.181867	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3968) [Reassembled in #15]
15	0.181867	172.26.45.151	193.136.9.240	ICMP	1064	Echo (ping) request id=0x0001, seq=10370/33320, ttl=2 (no response found!)
16	0.184269	172.16.2.1	172.26.45.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
17	0.231924	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=3969) [Reassembled in #19]
18	0.231924	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3969) [Reassembled in #19]
19	0.231924	172.26.45.151	193.136.9.240	ICMP	1064	Echo (ping) request id=0x0001, seq=10371/33576, ttl=3 (no response found!)
20	0.240601	172.16.115.252	172.26.45.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
21	0.281961	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=396a) [Reassembled in #23]
22	0.281961	172.26.45.151	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=396a) [Reassembled in #23]
23	0.281961	172.26.45.151	193.136.9.240	ICMP	1064	Echo (ping) request id=0x0001, seq=10372/33832, ttl=4 (reply in 26)
24	0.342846	193.136.9.240	172.26.45.151	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=e1fb) [Reassembled in #26]
25	0.342846	193.136.9.240	172.26.45.151	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=e1fb) [Reassembled in #26]
26	0.342846	193.136.9.240	172.26.45.151	ICMP	1064	Echo (ping) reply id=0x0001, seq=10372/33832, ttl=61 (request in 23)
27	0.433457	172.26.45.151	142.250.200.138	UDP	75	61003 → 443 Len=33
28	0.458884	142.250.200.138	172.26.45.151	UDP	67	443 → 61003 Len=25
29	1.264821	172.26.45.151	142.250.200.138	UDP	75	61003 → 443 Len=33

Fig. 18 - Pacotes com Fragmentação

1.3 Exercício 3

1.3.1 Alínea a)



8	0.094280	193.136.9.240	172.26.45.151	ICMP	1064 Echo (ping) reply id=0x0001, seq=10368/32808, ttl=61 (request in 5)
9	0.130922	172.26.45.151	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=0, ID=3967) [Reassembled in #11]
10	0.130922	172.26.45.151	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=ICMP 1, off=1480, ID=3967) [Reassembled in #11]

Fig. 19 - Pacote 1 e Fragmentação

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

Tendo em conta que a **MTU** (*Maximum Transmission Unit*) que foi utilizada tinha capacidade para enviar pacotes com um tamanho máximo de apenas 1500 *bytes* e o nosso pacote possuía um tamanho de 4010 *bytes*, houve necessidade deste mesmo pacote ser fragmentado de forma que possibilitasse o seu envio.

1.3.2 Alínea b)

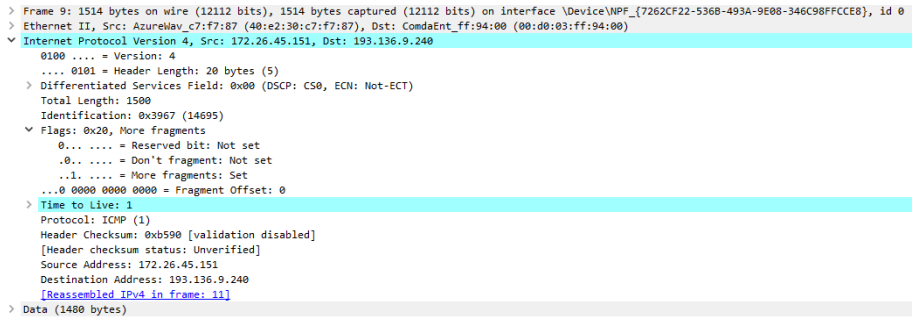


Fig. 20 - Fragmento 1 do Pacote 1

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?

O campo que nos permite averiguar se um pacote foi ou não fragmentado são as *flags*. Podemos identificar o primeiro fragmento através dos valores do *fragment offset* (que deve ter o valor **0**) e do *more fragments* (que deve ter o valor **1**, indicando a existência de mais fragmentos). Desta forma, e como podemos observar na figura 20, esta representa o primeiro fragmento. O tamanho deste datagrama IP é de 1500 *bytes*.

1.3.3 Alínea c)

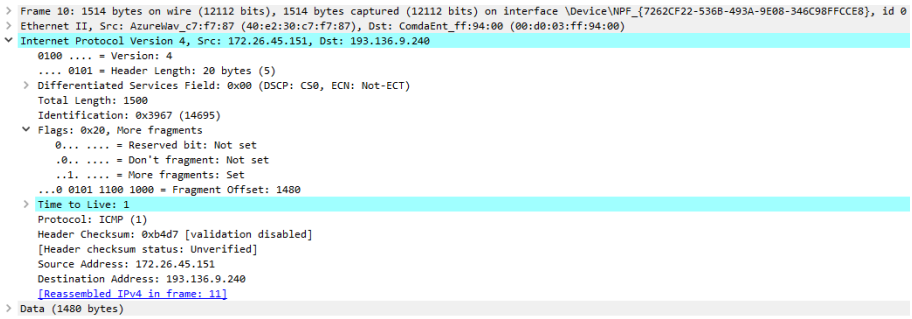


Fig. 21 - Fragmento 2 do Pacote 1

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?

O campo que nos indica que não se trata do 1º fragmento é, como foi explicado anteriormente, o *fragment offset* que, como podemos ver na figura 21, é diferente de **0**. Podemos também verificar o facto de que existem mais fragmentos, visto que o valor do *more fragments* é **1**.

1.3.4 Alínea d)

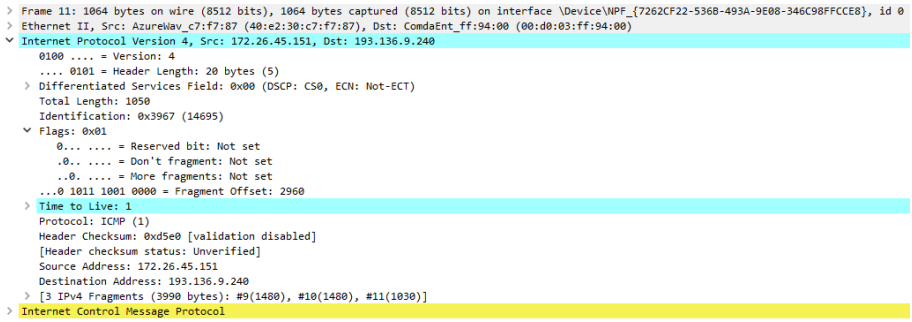


Fig. 22 - Fragmento 3 do Pacote 1

Quantos fragmentos foram criados a partir do datagrama original?

Podemos verificar se um fragmento é o último, mais uma vez, através das *flags*. Neste caso, o último fragmento terá um valor de **0** na *flag more fragments* e um valor diferente de **0** na *fragment offset*. Isto verifica-se na figura 22, pelo que podemos concluir que o número total de fragmentos criados é **3**.

1.3.5 Alínea e)

Indique, resumindo, os campos que mudam no cabeçalho IP entre os diferentes fragmentos, e explique a forma como essa informação permite reconstruir o datagrama original.

Os campos nos quais se verificam alterações nos diferentes fragmentos são, como referido anteriormente, as *flags*, nomeadamente o *fragment offset* e o *more fragments*. Estes têm as funções de nos indicar a ordem pela qual os fragmentos devem ser organizados e se existem ou não mais fragmentos, respetivamente. A ordem pela qual se organizam é especialmente importante para a reconstrução do datagrama original, tendo em conta que isto será possível se os fragmentos forem organizados por ordem crescente.

1.3.6 Alínea f)

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

Como referido anteriormente, o pacote original foi dividido em três fragmentos, sendo que dois deles têm um tamanho de 1500 *bytes* e um deles tem tamanho de 1010 *bytes*. No entanto, devemos retirar os valores dos cabeçalhos a dois destes fragmentos, de modo a obter o tamanho real do pacote original. Desta forma, aos 4010 *bytes* retiramos 40 *bytes* pertencentes aos cabeçalhos, sobrando então os **3070 *bytes*** que representam, de facto, o tamanho do pacote original. (1480 + 1480 + 1010 + 40 = 4010)

1.3.7 Alínea g)

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

De modo a possibilitar a deteção do último fragmento do datagrama original, devemos ter em atenção as duas *flags* referidas em perguntas anteriores: *more fragments* e *fragment offset*. Desta forma, as expressões que nos permitiriam ter sucesso neste processo seriam uma junção de (*fragment offset != 0*) e (*more fragments == 0*).

2 Parte 2

Considere que a topologia de rede *LEI-RC* é distribuída por quatro departamentos (A, B, C e D) e cada departamento possui um router de acesso à sua rede local. Estes routers de acesso (RA , RB , RC e RD) estão interligados entre si por ligações *Ethernet* a 1Gbps, formando um anel. Por sua vez, existe um servidor por departamento (SA , SB , SC , SD) e dois portáteis (pc) por departamento (A - *Bela, Monstro*; B - *Jasmine, Alladin*; C - *Ariel, Eric*; D - *Simba, Nala*), todos interligados ao *router* respetivo através de um comutador (*switch*). Cada servidor S tem uma ligação a 1Gbps e os *laptops* ligações a 100Mbps. Considere apenas a existência de um comutador por departamento. A conectividade IP externa da organização é assegurada através de um *router* de acesso *RISP* conectado a RA por uma ligação ponto-a-ponto a 1 Gbps. Construa uma topologia CORE que reflita a rede local da organização. Atribua as designações corretas aos equipamentos. Para facilitar a visualização pode ocultar o endereçamento IPv6. Grave a topologia para eventual reposição futura.

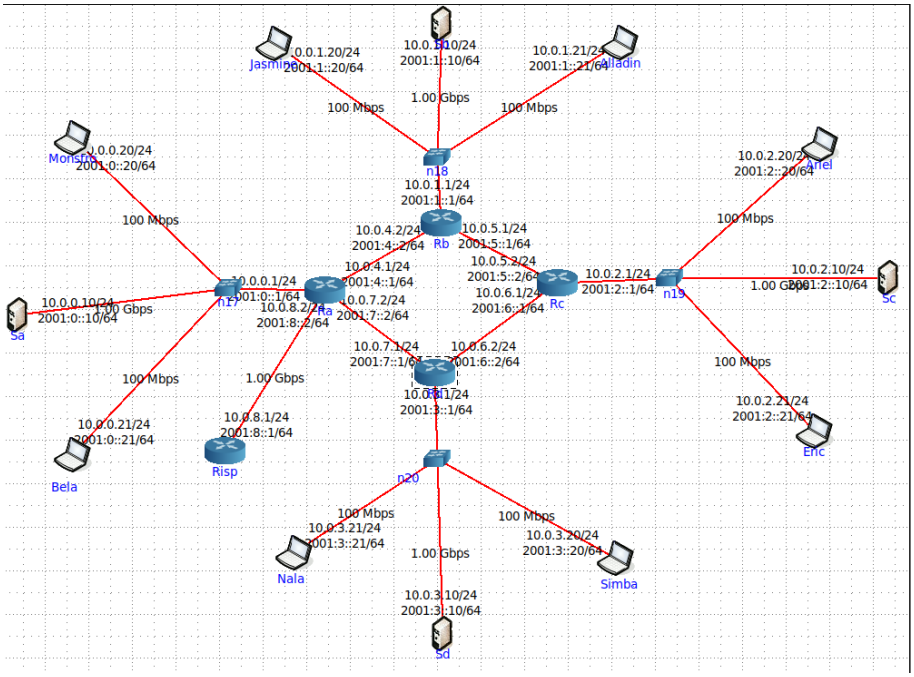


Fig. 23 - Topologia de rede LEI-RC

2.1 Exercício 1

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

2.1.1 Alínea 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.

A máscara usada foi **255.255.255** uma vez que todos os endereços apresentados na topologia terminam em **/24**. Podem verificar-se os endereços na figura 23.

2.1.2 Alínea b)

Tratam-se de endereços públicos ou privados? Porquê?

Uma vez que os endereços se encontram dentro da gama **10.0.0.0 - 10.255.255.255/8**, pode concluir-se que são endereços de IP privados, uma vez que todos os endereços apresentados começam por **10**.

2.1.3 Alínea c)

Porque razão não é atribuído um endereço IP aos switches?

Os *switches* atuam a nível de *hardware*, logo não conhecem o protocolo IP.

2.1.4 Alínea d)

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

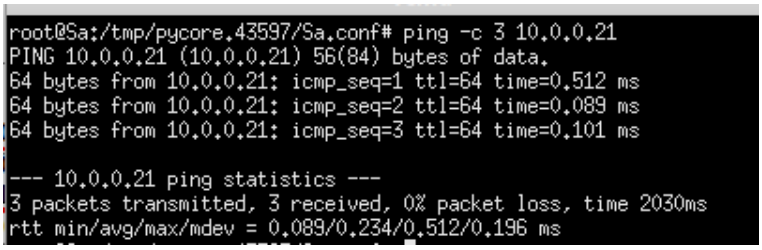


Fig. 24 - Rede A

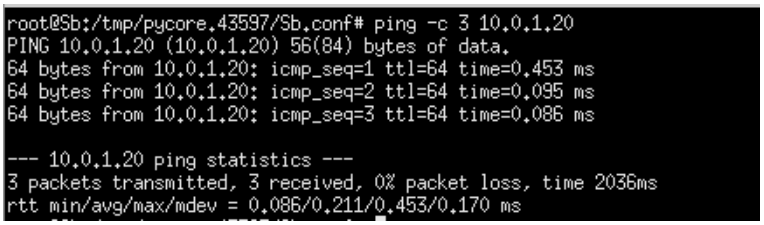


Fig. 25 - Rede B

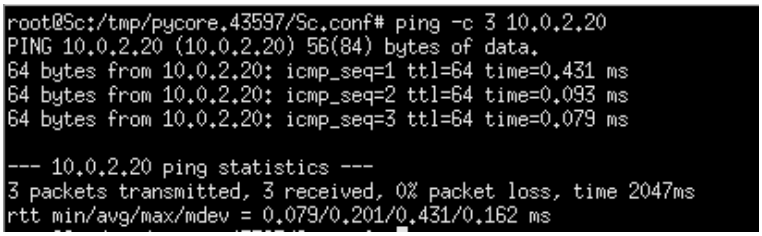


Fig 26 - Rede C

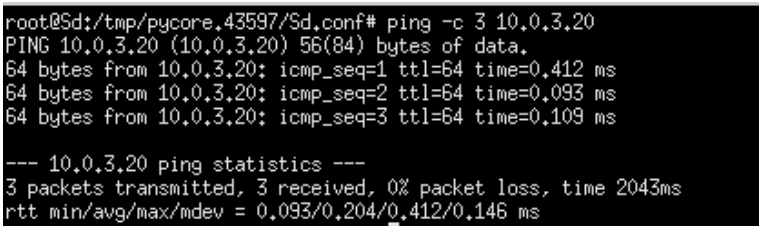


Fig. 27 - Rede D

Com as figuras acima conclui-se que existe conectividade entre dispositivos do mesmo departamento.

2.1.5 Alínea e)

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

Comando a partir do *host Alladin* para os outros departamentos:

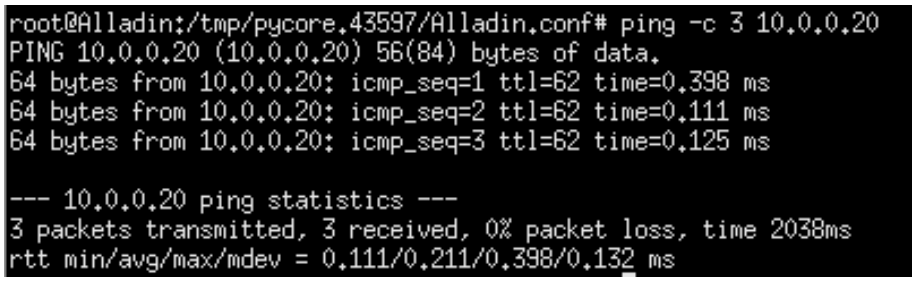


Fig. 28 - Rede A

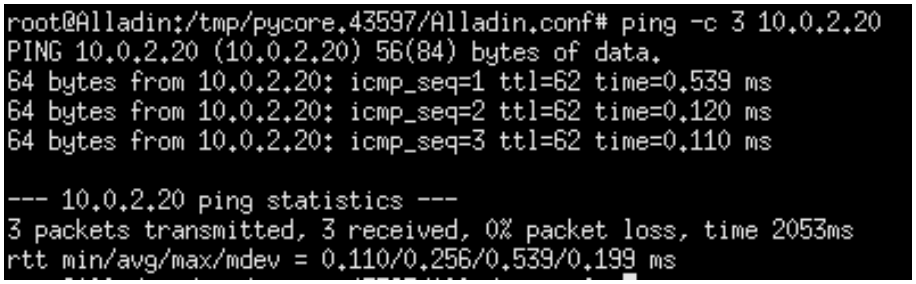


Fig. 29 - Rede C

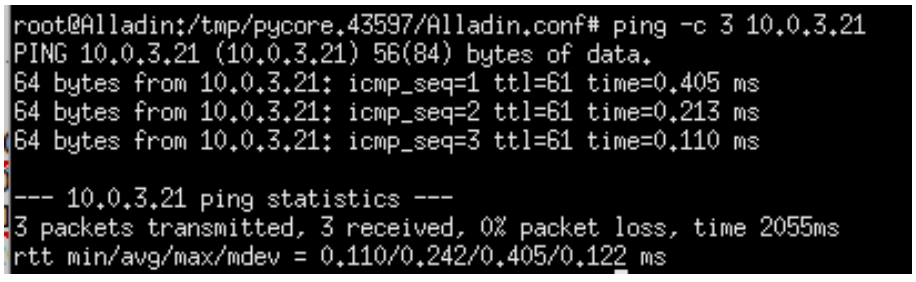


Fig. 30 - Rede D

2.1.6 Alínea f)

Verifique se existe conectividade IP do portátil Bela para o router de acesso R ISP.

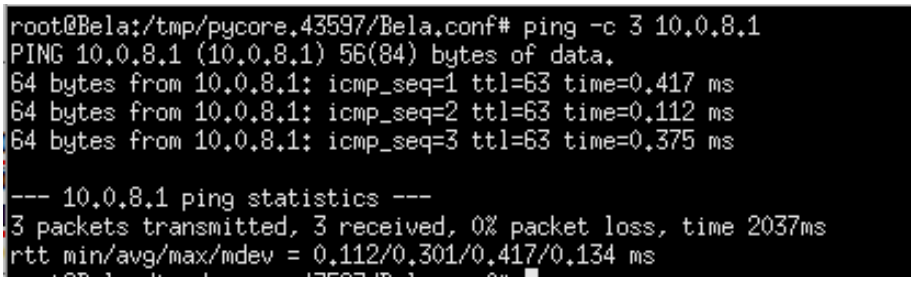


Fig. 31

Com a execução do comando ping a partir do portátil *Bela* com o IP do *router ISP* conclui-se que existe conctividade entre o portátil *Bela* e o *router ISP*.

2.2 Exercicio 2

2.2.1 Alínea a)

Execute o comando netstat –rn por forma a poder consultar a tabela de encaminhamento unicast (IPv4). Inclua no seu relatório as tabelas de encaminhamento obtidas; interprete as várias entradas de cada tabela. Se necessário, consulte o manual respetivo (man netstat).

```
root@Ra:/tmp/pycore.43597/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        10.0.4.2        255.255.255.0   UG        0 0          0 eth1
10.0.2.0        10.0.4.2        255.255.255.0   UG        0 0          0 eth1
10.0.3.0        10.0.7.1        255.255.255.0   UG        0 0          0 eth2
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth1
10.0.5.0        10.0.4.2        255.255.255.0   UG        0 0          0 eth1
10.0.6.0        10.0.7.1        255.255.255.0   UG        0 0          0 eth2
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
```

Fig. 32 - Tabela de Encaminhamento do Router A

```
root@Bela:/tmp/pycore.43597/Bela.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.0.1        0.0.0.0         UG        0 0          0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

Fig. 33 - Tabela de encaminhamento do portátil Bela

No que diz respeito à figura 32 (Tabela de encaminhamento do Router A) temos que para qualquer rede de destino existe um *Gateway* para onde o tráfego deve ser direcionado. Contudo, em algumas linhas, verifica-se que o *Gateway* é **0.0.0.0.** Isto significa que não existe um *Gateway* para aquele destino em concreto. Este facto é comprovado com a falta da *flag G* na mesma linha. O que acontece nestes casos é que com a ajuda do protocolo **ARP** se encontra o endereço **MAC** da máquina destino e assim se envia diretamente o tráfego.

Quanto à figura 33 (Tabela de encaminhamento do portátil *Bela*), como dá para verificar na imagem correspondente, só há 2 linhas. Uma delas representa o encaminhamento do tráfego para fora da sua sub-rede e a outra representa o tráfego para a rede local.

2.2.2 Alínea b)

Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema, por exemplo, ps -ax ou equivalente).

```
root@Ra:/tmp/pycore.40621/Ra.conf# ps -ax
    PID TTY      STAT   TIME COMMAND
      1 ?        S      0:00 vnoded -v -c /tmp/pycore.40621/Ra -l /tmp/pycore.40
     75 ?        Ss     0:00 /usr/local/sbin/zebra -d
     81 ?        Ss     0:00 /usr/local/sbin/ospf6d -d
     86 ?        Ss     0:00 /usr/local/sbin/ospfd -d
     93 pts/2    Ss     0:00 /bin/bash
    100 pts/2    R+     0:00 ps -ax
```

Fig. 34 - Execução de ps -ax no Router A

```
root@Bela:/tmp/pycore.40621/Bela.conf# ps -ax
    PID TTY      STAT   TIME COMMAND
      1 ?        S      0:00 vnoded -v -c /tmp/pycore.40621/Bela -l /tmp/pycore.
     20 pts/2    Ss     0:00 /bin/bash
     27 pts/2    R+     0:00 ps -ax
```

Fig. 35 - Execução de ps -ax no portátil Bela

Ao analisar os processos em execução nas diferentes máquinas, conclui-se que os *routers* estão a usar encaminhamento dinâmico e os portáteis e os servidores usam encaminhamento estático, como se pode concluir nas figuras 34 e 35, representadas acima.

2.2.3 Alínea c)

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

```
root@Sa:/tmp/pycore.46045/Sa.conf# route delete default
root@Sa:/tmp/pycore.46045/Sa.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
```

Fig. 36

A rota *default* era a única que o Servidor Sa conhecia para comunicar com redes fora a sua, resultando este comando num impedimento de transmissão de informação para fora da sua rede (Rede A). Isto significa que apenas os portáteis e o *router* presentes na Rede A conseguem comunicar com o Servidor Sa. Segue-se uma tentativa de acesso de um portátil da Rede C (*Ariel*) ao servidor Sa:

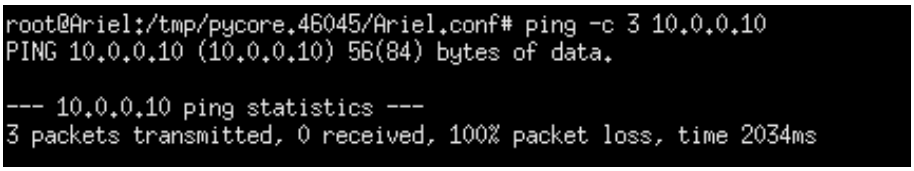


Fig. 37

Como podemos verificar, o portátil da Rede C não consegue comunicar com o servidor Sa.

2.2.4 Alínea 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.

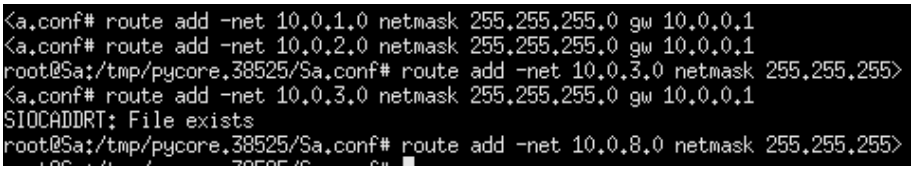


Fig. 38

Comandos utilizados:

- **route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.0.1**
- **route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.0.1**
- **route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.0.1**
- **route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.0.1**

Com estes comandos adicionamos à *route* os departamentos das redes B, C e D (10.0.1.0, 10.0.2.0 e 10.0.3.0 respetivamente) e o *RISP* (10.0.8.0)

2.2.5 Alínea e)

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

```
root@Sb:/tmp/pycore.38525/Sb.conf# ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=63 time=0.215 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=63 time=0.088 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=63 time=0.094 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2056ms
rtt min/avg/max/mdev = 0.088/0.132/0.215/0.058 ms
```

Fig. 39 - Ping entre Servidor Sb e Servidor Sa

```
root@Sc:/tmp/pycore.38525/Sc.conf# ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=62 time=0.207 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=62 time=0.079 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=62 time=0.101 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2055ms
rtt min/avg/max/mdev = 0.079/0.129/0.207/0.055 ms
```

Fig. 40 - Ping entre Servidor Sc e Servidor Sa

```
root@Sd:/tmp/pycore.38525/Sd.conf# ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=63 time=0.367 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=63 time=0.130 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=63 time=0.103 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2047ms
rtt min/avg/max/mdev = 0.103/0.200/0.367/0.118 ms
```

Fig. 41 - Ping entre Servidor Sd e Servidor Sa

```
root@Risp:/tmp/pycore.38525/Risp.conf# ping -c 3 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.454 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.079 ms
64 bytes from 10.0.0.1: icmp_seq=3 ttl=64 time=0.101 ms

--- 10.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2033ms
rtt min/avg/max/mdev = 0.079/0.211/0.454/0.171 ms
```

Fig. 42 - Ping entre Risp e Servidor Sa

2.3 Exercício 3

2.3.1 Alínea 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 (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.

Tendo em conta as instruções que nos foram dadas, nosso IP será dado por 192.168.10.128/25. Tendo em conta o nosso objetivo que é, neste caso, criar 4 sub-redes, devemos utilizar **3 *bits*** (sendo que com 2 *bits* poderiamos definir apenas 2 sub-redes), o que nos dá então a possibilidade de crirar 8 sub-redes, das quais duas deixaremos reservadas. A utilização de 3 *bits* envolve também o incremento da máscara de 25 para 28.

000	Reservado	-
001	Livre	Departamento A
010	Livre	Departamento B
011	Livre	
100	Livre	Departamento C
101	Livre	
110	Livre	Departamento D
111	Reservado	-

2.3.2 Alínea 2)

2.3.3 Alínea 3)

3 Conclusão

Numa primeira fase do trabalho foi-nos pedida uma análise do protocolo IPv4, para a qual utilizamos a topologia CORE de forma que nos capacitasse a análise dos datagramas e do tráfego ICMP. Isto contribuiu para um melhor entendimento do processo de transmissão de dados entre variadas máquinas ligadas à mesma rede.

Quanto à segunda parte do projeto, podemos dizer que esta se focou mais no processo de endereçamento e encaminhamento de IP. Nesta fase foi também necessário criar vários departamentos (novamente com a ajuda do CORE) de modo a averiguar a possibilidade de conexões entre si, bem como a forma como eram feitas estas conexões.

No geral consideramos que este projeto foi bastante interessante e importante para adquirir mais conhecimento em relação aos conceitos abordados, bem como para pôr em prática aquilo que já sabíamos.