Universidade do Minho

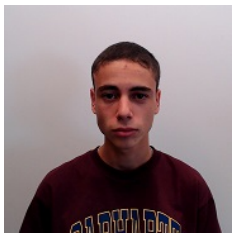
Licenciatura em Engenharia Informática

Redes de Computadores

Trabalho Prático 2

Grupo 34

Diogo Rebelo
(A93278)



Hugo Brandão
(A93287)

Gonçalo Freitas
(A93297)

20 de junho de 2023

Conteúdo

1. Questões e Respostas - Parte I

Questão 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.

Começou-se por construir toda a topologia no CORE, seguindo as instruções e metodologia referidas no enunciado da respetiva questão. Foram, então, criadas várias estruturas de rede, nomeadamente:

- dois hosts (pc e servidor), designados Bela e Monstro, respetivamente;
- quatro routers (R1, R2, R3, R4 e R5), com delays de ligação de 10 ms.

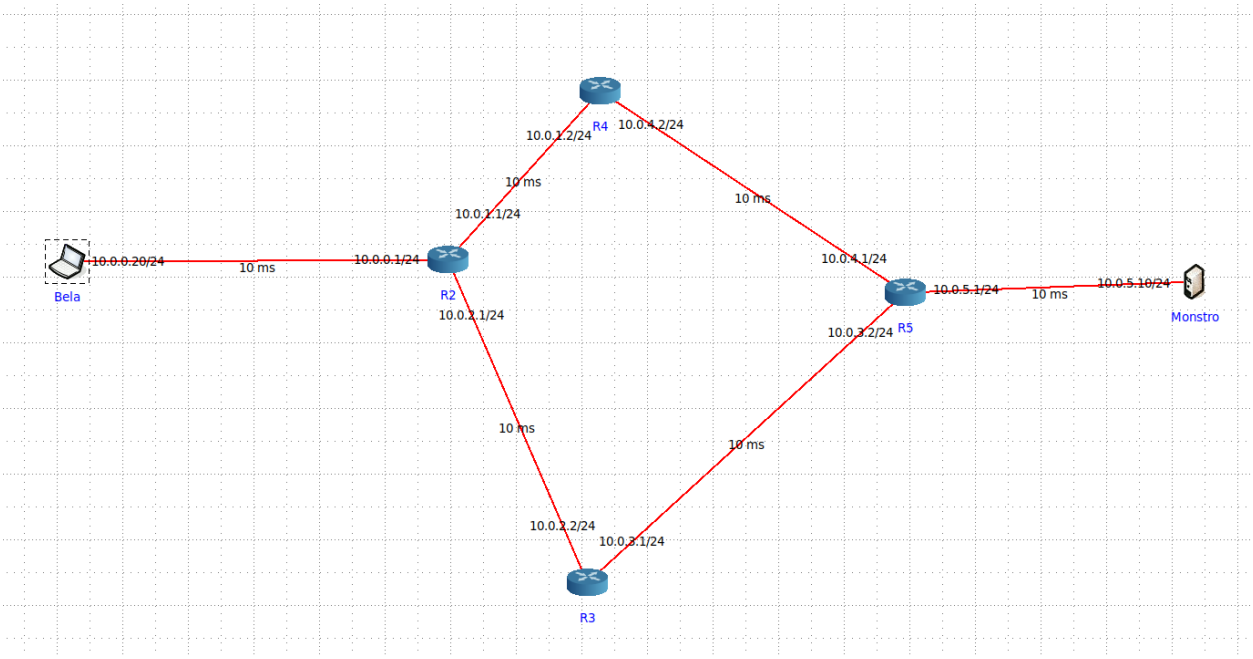
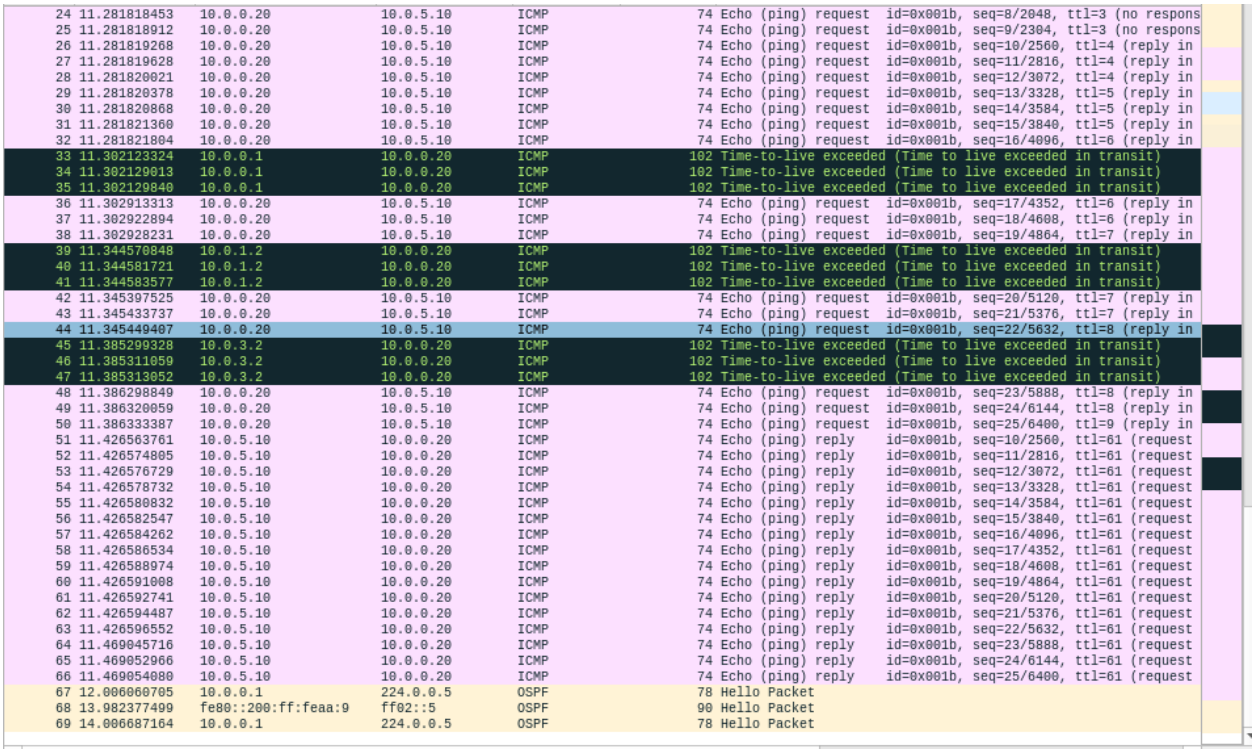


Figura 1: Topologia construída, segundo as instruções supracitadas.

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.

Após ativar a topologia, esperou-se cerca de 90 segundos para que houvesse conectividade entre a Bela e o Monstro, já que este intervalo de tempo inicial é normalmente usado para o reconhecimento da topologia. De seguida, iniciou-se a captura de tráfego no Wireshark, a partir do host Bela, obtendo-se a seguinte captura:



24	11.281818453	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=8/2048, ttl=3 (no respons
25	11.281818912	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=9/2304, ttl=3 (no respons
26	11.281819268	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=10/2560, ttl=4 (reply in
27	11.281819628	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=11/2816, ttl=4 (reply in
28	11.281820021	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=12/3072, ttl=4 (reply in
29	11.281820378	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=13/3328, ttl=5 (reply in
30	11.281820868	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=14/3584, ttl=5 (reply in
31	11.281821360	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=15/3840, ttl=5 (reply in
32	11.281821804	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=16/4096, ttl=6 (reply in
33	11.302123324	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
34	11.302129013	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
35	11.302129840	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
36	11.302913313	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=17/4352, ttl=6 (reply in
37	11.302922894	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=18/4608, ttl=6 (reply in
38	11.302928231	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=19/4864, ttl=7 (reply in
39	11.344570848	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
40	11.344581721	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
41	11.344583577	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
42	11.345397525	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=20/5120, ttl=7 (reply in
43	11.345433737	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=21/5376, ttl=7 (reply in
44	11.345449407	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=22/5632, ttl=8 (reply in
45	11.385299328	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
46	11.385311059	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
47	11.385313052	10.0.3.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
48	11.386298849	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=23/5888, ttl=8 (reply in
49	11.386320059	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=24/6144, ttl=8 (reply in
50	11.386333387	10.0.0.20	10.0.5.10	ICMP	74 Echo (ping) request id=0x001b, seq=25/6400, ttl=9 (reply in
51	11.426563761	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=10/2560, ttl=61 (request
52	11.426574805	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=11/2816, ttl=61 (request
53	11.426576729	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=12/3072, ttl=61 (request
54	11.426578732	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=13/3328, ttl=61 (request
55	11.426580832	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=14/3584, ttl=61 (request
56	11.426582547	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=15/3840, ttl=61 (request
57	11.426584262	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=16/4096, ttl=61 (request
58	11.426586534	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=17/4352, ttl=61 (request
59	11.426588974	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=18/4608, ttl=61 (request
60	11.426591008	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=19/4864, ttl=61 (request
61	11.426592741	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=20/5120, ttl=61 (request
62	11.426594487	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=21/5376, ttl=61 (request
63	11.426596552	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=22/5632, ttl=61 (request
64	11.469045716	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=23/5888, ttl=61 (request
65	11.469052966	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=24/6144, ttl=61 (request
66	11.469054080	10.0.5.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x001b, seq=25/6400, ttl=61 (request
67	12.006060705	10.0.0.1	224.0.0.5	OSPF	78 Hello Packet
68	13.982377499	fe80::200:ff:feaa:9	ff02::5	OSPF	90 Hello Packet
69	14.006687164	10.0.0.1	224.0.0.5	OSPF	78 Hello Packet

Figura 2: Captura de tráfego no Wireshark

De seguida, no terminal, efetuou-se o comando, tendo-se obtido o output seguinte relativo ao rastreio de rotas:

```
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)  21.098 ms  21.049 ms  21.036 ms  21.025 ms  21.016 ms
 2  10.0.2.2 (10.0.2.2)  41.827 ms  41.818 ms  41.808 ms  41.799 ms  41.790 ms
 3  10.0.3.2 (10.0.3.2)  62.277 ms  62.269 ms  62.260 ms  62.251 ms  62.242 ms
 4  10.0.5.10 (10.0.5.10)  82.829 ms  82.668 ms  82.623 ms  82.609 ms  82.599 ms
root@Bela:/tmp/pycore.36281/Bela.conf#
```

Figura 3: Output do comando `traceroute -q 5 -I 10.0.5.10`

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.

Funcionamento do traceroute Antes de analisar a captura apresentada anteriormente, é relevante compreender o funcionamento do utilitário em questão. O traceroute é uma ferramenta que permite descobrir a rota entre dois hosts numa rede local ou remota, fazendo uso dos protocolos IP e ICMP e do campo TTL (time to live) presente no cabeçalho IP (IPv4) dos datagramas. Este TTL é, então, o número de saltos entre máquinas que os pacotes podem demorar numa rede, antes de serem descartados.

Todas as vezes que um datagrama chega a um router, é decrementado o valor do seu TTL. Isto permite que um determinado pacote fique em circulação infinitamente (caso haja alguma falha durante o roteamento). ENtão, quando um router recebe um datagrama com TTL igual a 0, ele já não encaminha esse pacote, descartando-o. É, de imediato, enviada uma mensagem (ICMP Time-to-live_exceeded), de volta ao host que originou o pacote. Um aspeto relevante é que essa mesma mensagem contém o endereço IP do router como endereço de origem, permitindo identificar cada router por onde o pacote vai passando. O número de mensagens ICMP Time-to-live exceeded, para TTLs diferentes, é igual a 3, traduzindo que o pacote passou por 3 routers entre estes dois hosts, no rastreio.

Análise de resultados Começam-se por enviar vários datagramas (mensagens ICMP do tipo Echo Request (ping)) com vários valores crescentes de TTL ao host de destino. Neste caso, em vez de se enviar um pacote de cada vez com TTLs crescentes, esperando por uma resposta entre cada pacote enviado, são enviados vários pacotes, sendo com base na resposta ICMP que se compreende quantos TTLs são necessários para se saber a rota dos datagramas entre os dois hosts. Ao chegar ao R1, o TTL do pacote (TTL = 1) é decrementado, ficando com o valor zero, de modo que o datagrama é descartado e a mensagem ICMP "Tempo Excedido" é enviada de volta à origem, identificando-se o R1. Depois, continuam-se a enviar mais mensagens de echo, mas surge uma nova mensagem ICMP "Tempo Excedido", desta vez com o TTL igual a 2. Esta mensagem passou pelo R1 (que decrementou o TTL para 1), chegando ao R2 (que também decrementou o TTL). Então, fica-se com TTL = 0, descobrindo-se o R2. Este processo continua até que um datagrama chegue ao host de destino, ou seja, até que na mensagem ICMP "Tempo Excedido" tenha um TTL = 4, já que na própria rede, o número de saltos mínimo entre os dois hosts é 4.

Relação com o output do comando no terminal Olhando para o output presente na figura 3, verifica-se que o número de saltos feitos pelo traceroute é 4, como se previa pela configuração da própria topologia. São também apresentados os 3 RTTs e os respetivos

endereços de IP. Também conseguimos perceber a rota que é feita:

$$10.0.0.20\,(host\,Bela) >> 10.0.0.1 >> 10.0.1.2 >> 10.0.3.2 >> 10.0.5.10\,(host\,Monstro)$$

Como referimos antes, no pacote ICMP de "Tempo excedido" o endereço de origem é o do router onde se descartou o pacote, o que se comprova na rota feita, que é a sequência destes endereços de origem.

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.

A figura 3 comprova o pretendido. Foram realizados 4 saltos entre os dois hosts, logo, é necessário que o TTL mínimo seja igual a 4 para que se alcance o servidor monstro. Pela prórpia topologia, isso é facilmente inferido. Considerando-se um TTL igual a 4, este e decrementado no R1, depois no R2, depois no R3 e finalmente no R5, sendo o seu valor final igual a 0 e chegando-se ao host Monstro.

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.

```
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)  21.098 ms  21.049 ms  21.036 ms  21.025 ms  21.016 ms
 2  10.0.2.2 (10.0.2.2)  41.827 ms  41.818 ms  41.808 ms  41.799 ms  41.790 ms
 3  10.0.3.2 (10.0.3.2)  62.277 ms  62.269 ms  62.260 ms  62.251 ms  62.242 ms
 4  10.0.5.10 (10.0.5.10)  82.829 ms  82.668 ms  82.623 ms  82.609 ms  82.599 ms
root@Bela:/tmp/pycore.36281/Bela.conf#
```

Figura 4: Output do comando `traceroute -q 5 -I 10.0.5.10`

Como sugerido, aumentou-se o número de pacotes enviados para 5 e o traceroute envia então 5 pacotes separados, por cada salto. É assim possível obter o valor médio do RTT, fazendo a média destes valores. Note-se que se pretende obter o tempo no acesso ao servidor, utilizando-se os valores do salto 4 (até ao salto 4):

$$\overline{RTT} = \frac{82.829 + 82.668 + 82.623 + 82.609 + 82.599}{5} = 82,6656\ ms$$

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?

O RTT é frequentemente usado como uma aproximação do atraso, mas como é uma soma dos atrasos de ida e volta, o atraso unidirecional real pode não ser estimado com precisão a partir deste. Ao calcular o OWD dividindo o RTT por 2, pressupõe-se que os caminhos de ida e volta são os mesmos em termos de congestionamento, número de saltos ou qualidade de serviço (QoS), o que normalmente não acontece. Assim sendo, a precisão da estimativa por RTT/2 depende da natureza da distribuição do atraso em ambas as direções: quanto mais simétricos são os atrasos nas duas direções, maior é a precisão. Para evitar esta má precisão, a medição pode ser feita através de um método direto que usa relógios sincronizados ou através de uma estimativa, seguindo uma fórmula específica de cálculo (função da distribuição de atraso da rede).

Questão 2

> Usando o wireshark capture o tráfego gerado pelo traceroute para os seguintes tamanhos de pacote: situação (i) sem especificar, i.e., usando o tamanho do pacote de prova por defeito; e situação (ii) (4000 + X) bytes, em que X é o número do grupo de trabalho. Utilize como máquina destino o host marco.uminho.pt. Pare a captura. Com base no tráfego capturado, identifique os pedidos ICMP Echo Request e o conjunto de mensagens devolvidas como resposta. Selecione a primeira mensagem ICMP capturada (referente à situação (i) tamanho por defeito) e centre a análise no nível protocolar IP (expanda o tab correspondente na janela de detalhe do wireshark).

Na máquina nativa, com a captura de tráfego ativa no wireshark, introduziu-se primeiramente o comando sugerido, `traceroute -I router-di.uminho.pt`, sem um tamanho de pacote especificado. Com a primeira mensagem ICMP selecionada, efetuou-se a captura de informações relativas ao protocolo IP:

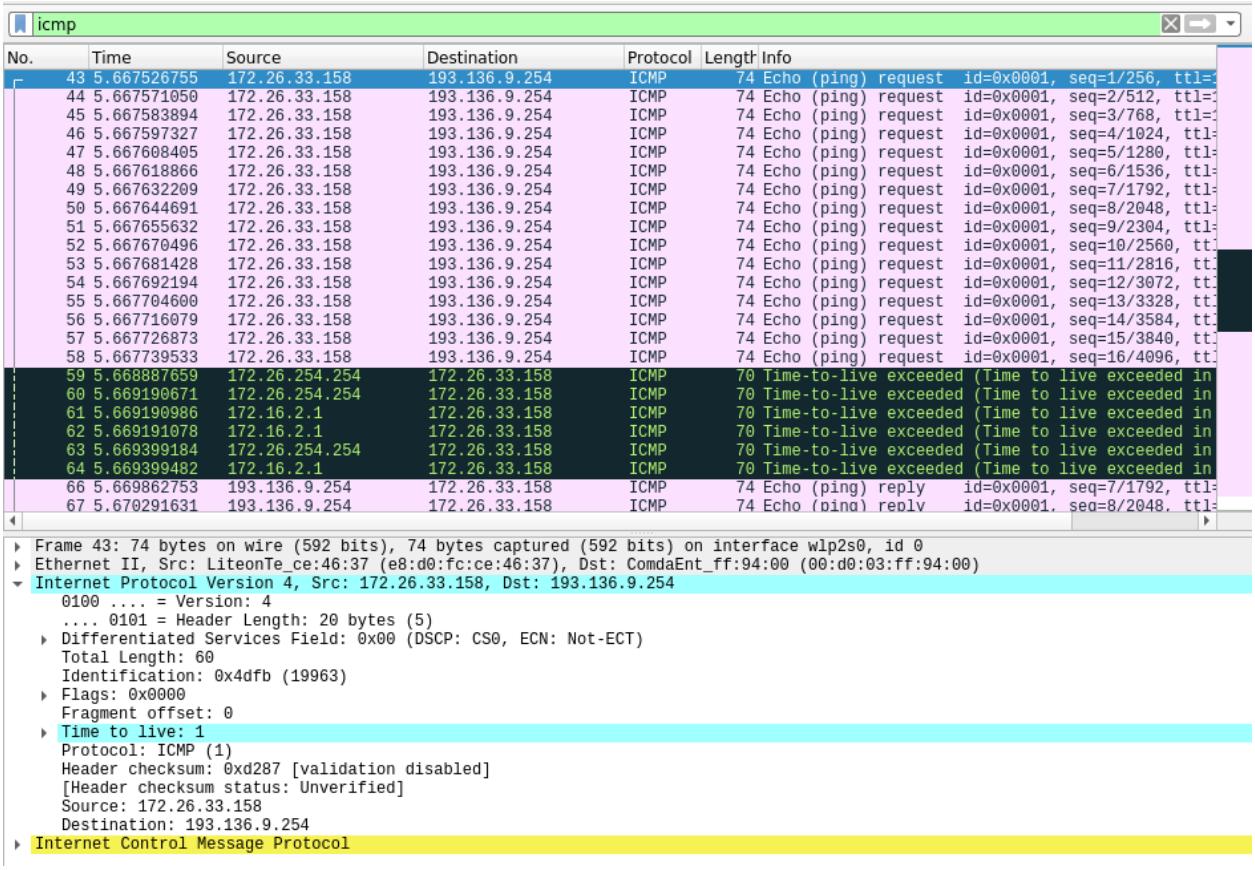


Figura 5: Informação do tráfego capturado com o primeiro comando.

Alínea a

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

Como se verifica na imagem acima, observando o IP de *Source*, o IP da interface ativa é 172.26.33.158

Alínea b

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

Como se verifica na imagem anterior, no campo *Protocol*, o valor do campo do protocolo é `ICMP (1)` [Hex: 0x01], permitindo informar que o protocolo encapsulado é o ICMP, sendo o número seguinte apenas um identificador padrão utilizado para mensagens ICMP.

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?

Também na imagem anterior, é possível observar nos campos *Header Length* e *Total Length*, que o tamanho do cabeçalho é de 20 bytes, com um tamanho total de 60 bytes, respetivamente. Então, tem-se um payload de 40 bytes, obtido subtraindo-se ao tamanho total do datagrama IP o tamanho do seu cabeçalho.

Alínea d

Expandindo a aba relativa às Flags, do pacote anterior, consegue-se observar que flags e offset estão a 0, percebendo-se se houve ou não fragmentação.

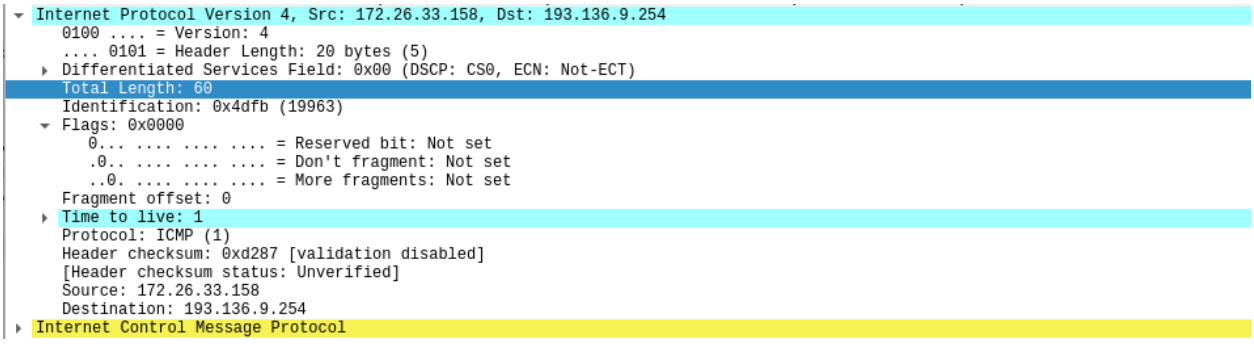


Figura 6: Informação relátiva às flags da mensagem ICMP selecionada.

Pela imagem anterior, o pacote não foi fragmentado, pois:

```
Fragment offset = 0
```

```
flags = 0
```

Alínea 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.

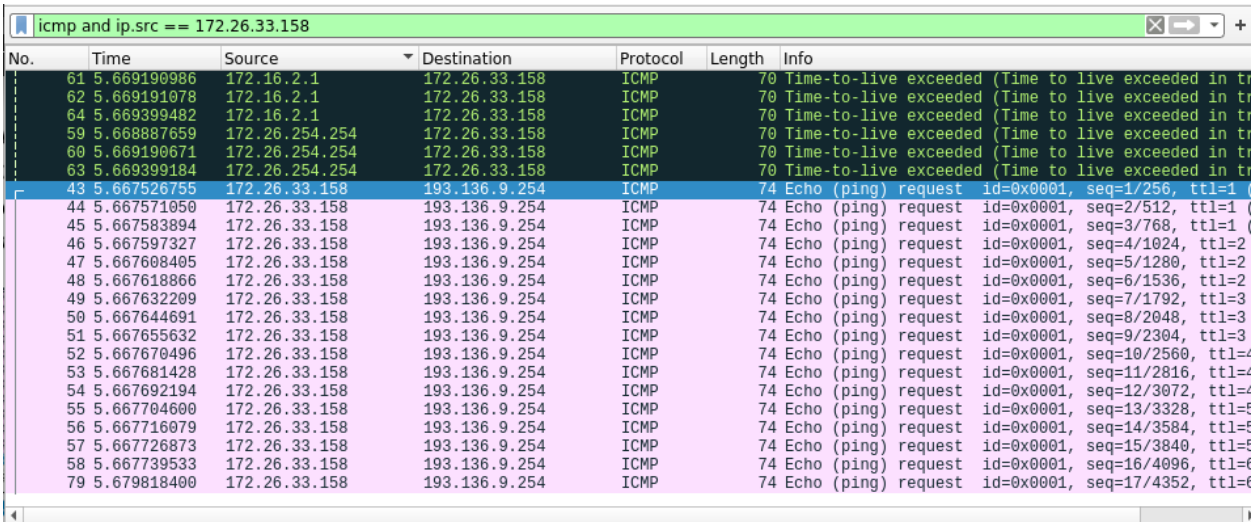


Figura 7: Captura de tráfego nas condições especificadas.

Na respetiva figura, facilmente se verifica que de entre cada conjunto de pacotes, há uma mudança no TTL (que vai sendo sucessivamente maior) e, apesar de existirem pacotes com o mesmo valor de TTL, este vai mudando a cada 3 pacotes. Selecionando-se dois pacotes (com diferentes valores de TTL), capturou-se a informação abaixo acerca deles:

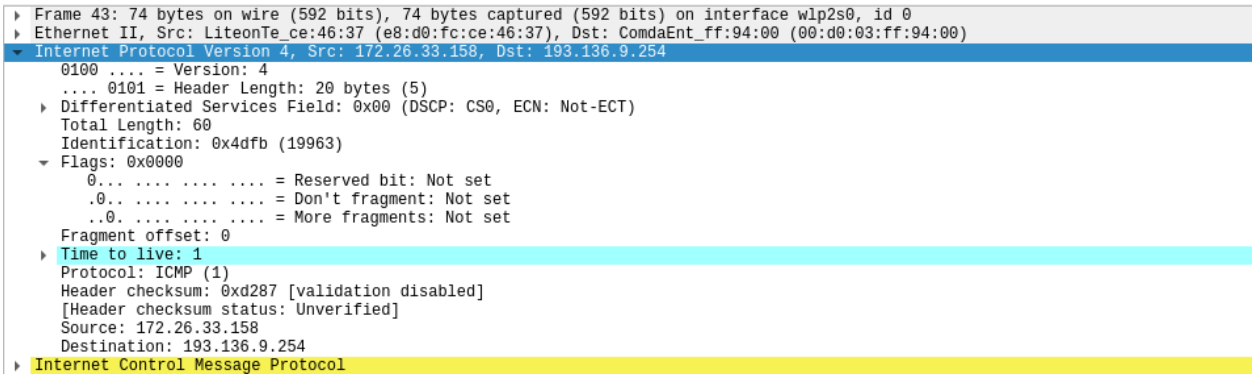


Figura 8: Informação sobre o pacote 1.

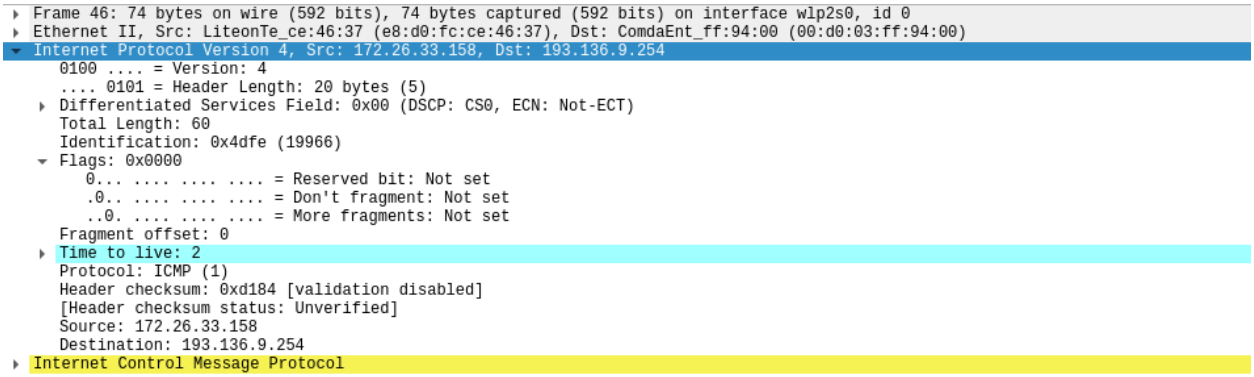


Figura 9: Informação sobre o pacote 2.

Como se comprova, de um pacote para o outro, há mudança nos valores de TTL. A *identification* também muda, assim como o *header checksum*.

Alínea f

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

Como se concluiu também na alínea anterior, são sempre enviados 3 pacotes com o mesmo TTL. Para além disso, o identificador de IP é incrementado em cada pacote. Para o efeito, selecionaram-se dois pacotes consecutivos, onde este incremento é visível.

Olhando para os valores de identificação do datagrama IP, nos pacotes abaixo, sendo estes consecutivos, o valor incrementa para um.

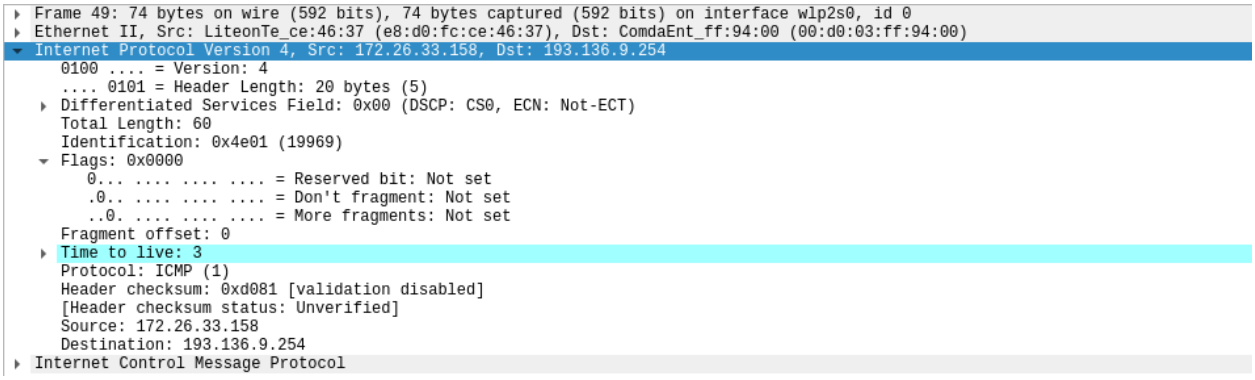


Figura 10: Informação sobre o pacote 1.

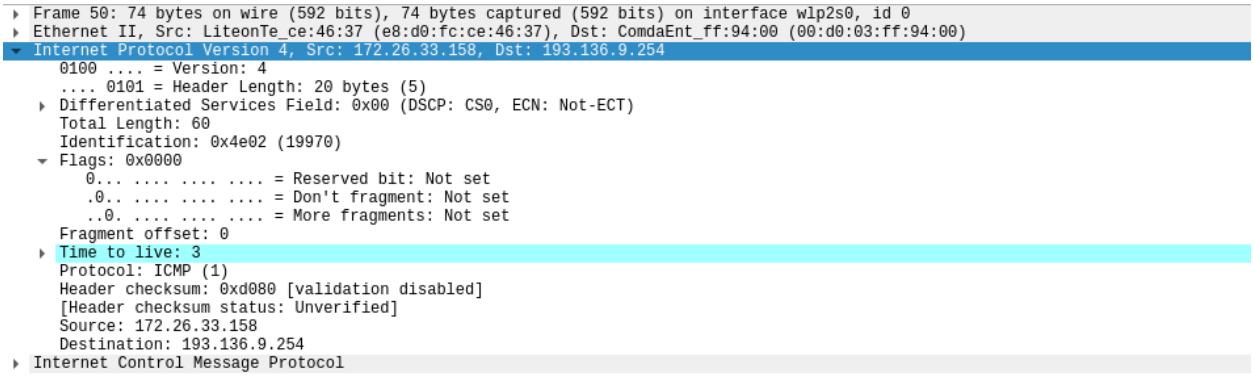


Figura 11: Informação sobre o pacote 2.

Uma observação relevante é que os valores, neste caso, em que não há fragmentação, têm de ter valores diferentes de identificação IP. Caso isso não acontecesse, significava que os pacotes com o mesmo valor de identificação IP seriam fragmentos de um mesmo datagrama IP.

Alínea 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?

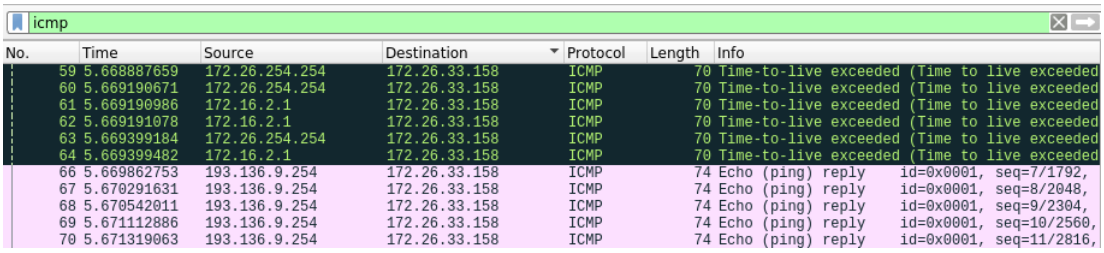


Figura 12: Captura de tráfego por ordenação com o Destino

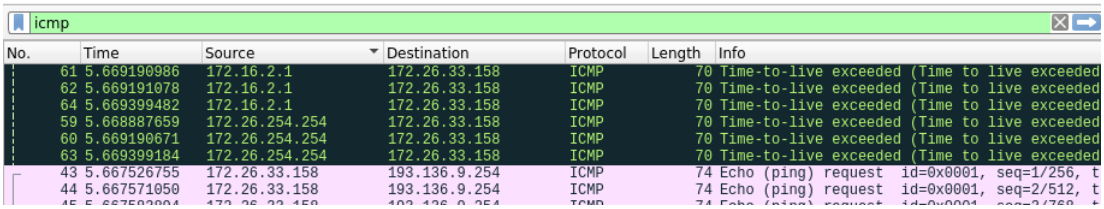


Figura 13: Captura de tráfego por ordenação com a Fonte

O valor do campo TTL varia. Para uma mesma *Source* (ordenando por esse parâmetro), verifica-se que os primeiros três datagramas têm o campo TTL a 255, os seguintes três datagramas têm este campo a 254. Isto acontece, porque são enviados três datagramas com um valor de TTL, outros três datagramas com o valor do TTL + 1 e assim sucessivamente até ao datagrama chegar ao destino. Por esta lógica, faz sentido que primeiro sejam recebidas mensagens ICMP com o valor do TTL a 254 e depois a 255.

```
     59 5.668887659    172.26.254.254        172.26.33.158         ICMP     70     Time-to-live exceeded
in transit)
Frame 59: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface wlp2s0, id 0
Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: LiteonTe_ce:46:37 (e8:d0:fc:ce:46:37)
Internet Protocol Version 4, Src: 172.26.254.254, Dst: 172.26.33.158
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 56
    Identification: 0xdd18 (56600)
    Flags: 0x0000
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..0. .... .... .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 255
    Protocol: ICMP (1)
    Header checksum: 0x651a [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.254.254
    Destination: 172.26.33.158
Internet Control Message Protocol
     61 5.669190986    172.16.2.1            172.26.33.158         ICMP     70     Time-to-live exceeded
in transit)
Frame 61: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface wlp2s0, id 0
Ethernet II, Src: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00), Dst: LiteonTe_ce:46:37 (e8:d0:fc:ce:46:37)
Internet Protocol Version 4, Src: 172.16.2.1, Dst: 172.26.33.158
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 56
    Identification: 0xdddc (56796)
    Flags: 0x0000
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..0. .... .... .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 254
    Protocol: ICMP (1)
    Header checksum: 0x631e [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.16.2.1
    Destination: 172.26.33.158
Internet Control Message Protocol
```

Figura 14: Informação sobre as mensagens ICMP “Tempo Excedido”.

Questão 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 de pacote ter sido definido para (4000 + X) bytes.

Iniciou-se a captura de tráfego e escreveu-se o comando que surge na imagem abaixo, especificando um novo tamanho (em bytes) para os respetivos pacotes.

```
purp@purp:~$ traceroute -I router-di.uminho.pt 4034
traceroute to router-di.uminho.pt (193.136.9.254), 30 hops max, 4034 byte packet
s
 1  _gateway (172.26.254.254)  2.854 ms  3.181 ms  3.672 ms
 2  172.16.2.1 (172.16.2.1)  3.113 ms  3.082 ms  3.577 ms
 3  router-di.uminho.pt (193.136.9.254)  8.828 ms  13.614 ms  18.237 ms
purp@purp:~$
```

Figura 15: Comando traceroute com especificação de tamanho 4000 + 34 = 4034.

Alínea a

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

Analisando a informação relativa à primeira mensagem ICMP Echo (ping) Request, vemos que `Frame = 25`. Compreende-se a necessidade de fragmentação já que

$$Length(packet) > MTU \Leftrightarrow 4034 > 1500$$

Então, como o valor da Unidade de Transmissão Máxima é maior que o valor do tamanho do pacote, há fragmentação. Para além disso, o *Fragment offset* é 2960 (diferente de zero) e observa-se que o pacote foi fragmentado em vários datagramas IP (informação do campo `3 IPv4 Fragments`, sugerido pelo software):

```
▾ [3 IPv4 Fragments (4014 bytes): #23(1480), #24(1480), #25(1054)]
    [Frame: 23, payload: 0-1479 (1480 bytes)]
    [Frame: 24, payload: 1480-2959 (1480 bytes)]
    [Frame: 25, payload: 2960-4013 (1054 bytes)]
    [Fragment count: 3]
    [Reassembled IPv4 length: 4014]
    [Reassembled IPv4 data: 08001851000300014849 4a4b4c4d4e4f5051525354555657…]
```

Figura 16: Detalhes do campo dos fragmentos.

Alínea 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?

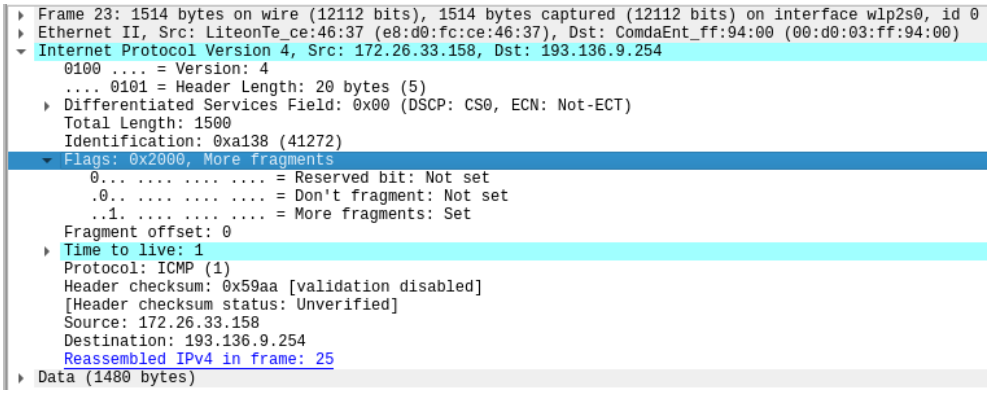


Figura 17: Detalhes do primeiro fragmento

Olhando para os valores das Flags, sabemos que o pacote possui mais fragmentos, pois tem o campo *More Fragments* a 1 e que o pacote em questão é o primeiro fragmento, pois o *Fragment Offset* está a 0. Este datagrama tem 1500 bytes (incluindo o header).

Alínea 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?

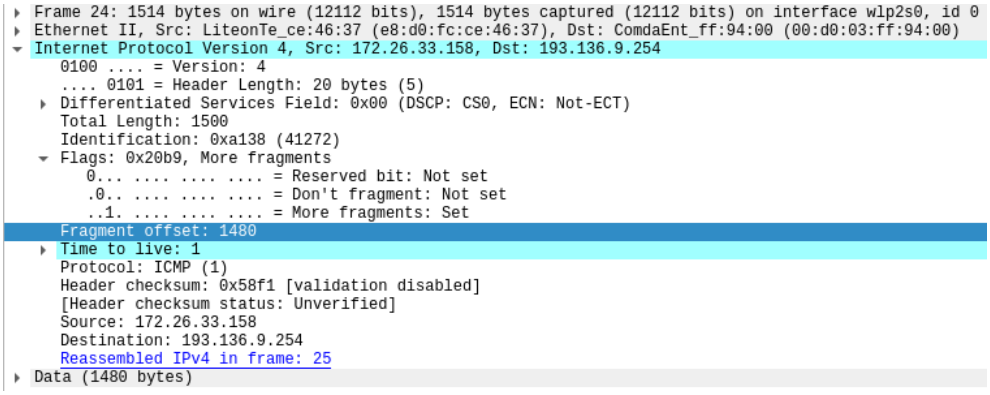


Figura 18: Detalhes do segundo fragmento

Podemos afirmar que não se trata do primeiro fragmento pois o *Fragment offset* é 1480 (diferente de 0) e que existem mais fragmentos já que o valor do *More fragments* é igual a 1 (diferente de 0).

Alínea d

> Quantos fragmentos foram criados a partir do datagrama original?

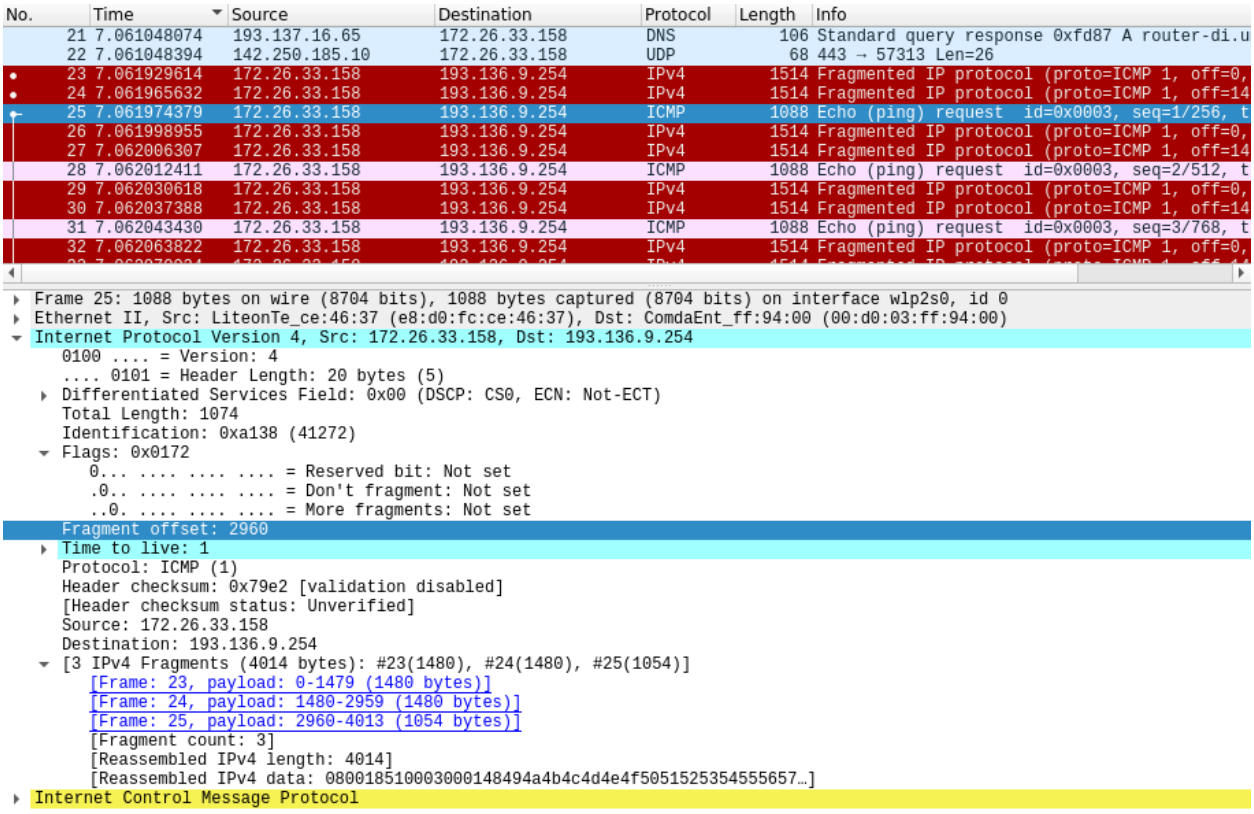


Figura 19: Número de Fragmentos do pacote inicial ICMP.

Como já se concluiu anteriormente, foram criados 3 (= *Fragment count*) fragmentos (com os `Frames` 23, 24 e 25) para o datagrama original. Em relação ao último fragmento (`Frame = 25`), este é detetado pois tem o campo *Fragment offset* diferente de 0 e o *More fragments* = 0. Os outros fragmentos (`Frame = 23` e `Frame = 24`) têm o *More fragments* = 1. Apesar de o *Wireshark* já nos informar do número de fragmentos nos detalhes do datagrama acima selecionado, uma forma de o concluir é olhar para o valor das flags, como se fez agora. Abaixo temos destacadas estas informações sobre as Flags.

```
Frame 23: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp2s0, id 0
Ethernet II, Src: LiteonTe_ce:46:37 (e8:d0:fc:ce:46:37), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.33.158, Dst: 193.136.9.254
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0xa138 (41272)
    Flags: 0x2000, More fragments
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..1. .... .... .... = More fragments: Set
    Fragment offset: 0
    Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x59aa [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.33.158
    Destination: 193.136.9.254
    Reassembled IPv4 in frame: 25
Frame 24: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface wlp2s0, id 0
Ethernet II, Src: LiteonTe_ce:46:37 (e8:d0:fc:ce:46:37), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.33.158, Dst: 193.136.9.254
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1500
    Identification: 0xa138 (41272)
    Flags: 0x20b9, More fragments
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..1. .... .... .... = More fragments: Set
    Fragment offset: 1480
    Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x58f1 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.33.158
    Destination: 193.136.9.254
    Reassembled IPv4 in frame: 25
Frame 25: 1088 bytes on wire (8704 bits), 1088 bytes captured (8704 bits) on interface wlp2s0, id 0
Ethernet II, Src: LiteonTe_ce:46:37 (e8:d0:fc:ce:46:37), Dst: ComdaEnt_ff:94:00 (00:d0:03:ff:94:00)
Internet Protocol Version 4, Src: 172.26.33.158, Dst: 193.136.9.254
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 1074
    Identification: 0xa138 (41272)
    Flags: 0x0172
        0... .... .... .... = Reserved bit: Not set
        .0.. .... .... .... = Don't fragment: Not set
        ..0. .... .... .... = More fragments: Not set
    Fragment offset: 2960
    Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x79e2 [validation disabled]
    [Header checksum status: Unverified]
    Source: 172.26.33.158
    Destination: 193.136.9.254
    [3 IPv4 Fragments (4014 bytes): #23(1480), #24(1480), #25(1054)]
        [Frame: 23, payload: 0-1479 (1480 bytes)]
        [Frame: 24, payload: 1480-2959 (1480 bytes)]
        [Frame: 25, payload: 2960-4013 (1054 bytes)]
        [Fragment count: 3]
        [Reassembled IPv4 length: 4014]
        [Reassembled IPv4 data: 08001851000300014849 4a4b4c4d4e4f5051525354555657…]
Internet Control Message Protocol
```

Figura 20: Informação relativa aos fragmentos do datagrama original.

O valor da identificação de cada datagrama para fragmentos é exatamente o mesmo, permitindo também comprovar o número de fragmentos supracitado.

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 do cabeçalho IP que mudam entre os vários fragmentos são: *total length, flags, fragment offset* e *checksum*.

Entre os dois primeiros pacotes e o último pacote, há uma mudança no *total length* (o qual passa de 1500 bytes para 1074 bytes) e também nas *Flags* (os dois primeiros fragmentos têm o bit *More fragments* a 1 e o último tem-no a 0). Há uma mudança também no valor do *checksum* em cada fragmento. Finalmente, o *Fragment offset* é igual a 0 para o primeiro fragmento e diferente de 0 nos outros. Este último campo e as flags são úteis para a reconstrução do datagrama original e permitem fazê-lo já que identificam a ordem de cada fragmento. Olhando para estes campos, podemos reconstruir o datagrama original:

Fragmento (Frame)	**More Fragments**	**Fragment offset**	**Ordem dos fragmentos**
23	1 =>indica que existem mais fragmentos	0 =>indica que é o primeiro fragmento	1º
24	1 =>indica que existem mais fragmentos	1480 (diferente de 0)	2º
25	0 =>indica que é o último	2960 (diferente de 0)	3º
Identifica o frame	Indica se existem ou não mais fragmentos	Indica a ordem dos fragmentos: 0 <1480 <2960	Estes fragmentos formam o datagrama original

Tabela 1: Interpretação e Reconstrução do Datagrama.

Alínea f

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

Sabemos que o tamanho definido para cada pacote foi de 4034 bytes. Assim, sendo o header de 20 bytes, temos um payload de 4034 - 20 = 4014. Para verificar esta fragmentação, basta somar os payloads de cada fragmento e verificar se é igual ao do datagrama original. Seja p_o o payload do datagrama original e p_{fi} o payload dos diferentes fragmentos (com i = 1,2,...). Assim, tem de verificar-se:

$$p_o = p_{f1} + p_{f2} + p_{f3} \Leftrightarrow$$
$$\Leftrightarrow 4014 = (1500 - 20) + (1500 - 20) + (1074 - 20) \Leftrightarrow$$

$$\Leftrightarrow 4014 = (1480) * 2 + 1054 \Leftrightarrow$$

$$\Leftrightarrow 4014 = 4014 \Leftrightarrow$$

$$True$$

Alínea g

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

$more\_fragments = 0$ && $fragment\_offset \neq 0$

A expressão anterior identifica o último fragmento do datagrama original. Contudo, a tabela presente na alínea e) anterior permite também concluir que no último fragmento a flag *More fragments* é 0 e o *Fragment offset* é diferente de 0 e tem o maior valor em relação aos outros fragmentos.

2. Questões e Respostas - Parte II

Questão 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.

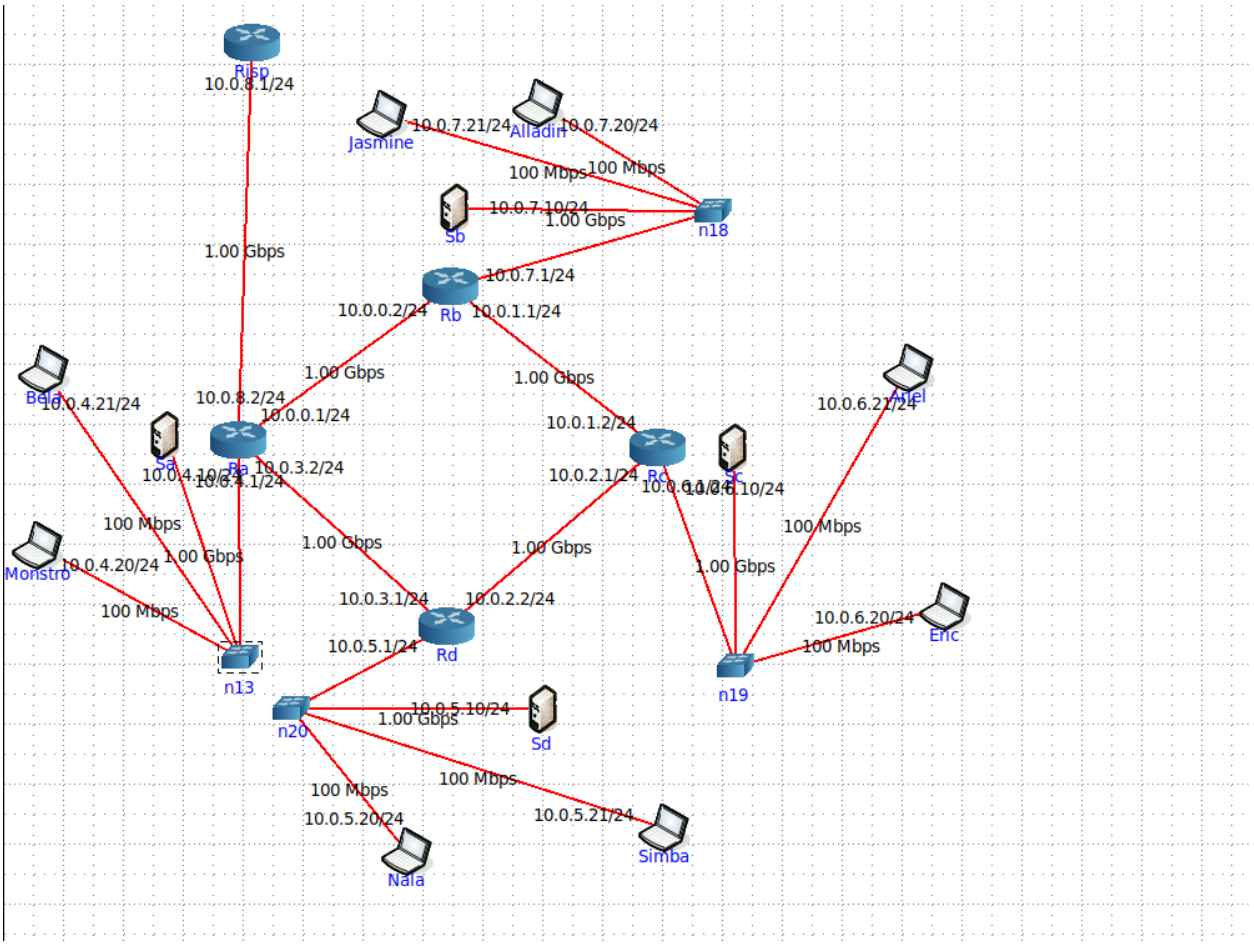


Figura 21: Topologia

A máscara de rede atribuída foi igual para todos os equipamentos "/24".
Tendo em conta que a imagem pode parecer um pouco confusa, deixamos abaixo uma lista de todos os os IP's:

- Bela: 10.0.4.21
- Monstro: 10.0.4.20
- Sa: 10.0.4.10
- Jasmine: 10.0.7.21
- Alladin: 10.0.7.20
- Sb: 10.0.7.10

- Eric: 10.0.6.20
- Ariel: 10.0.6.21
- Sc: 10.0.6.10
- Naia: 10.0.5.20
- Simba: 10.0.5.21
- Sd: 10.0.5.24
- Risp: 10.0.8.1

Alínea b

> Tratam-se de endereços públicos ou privados? Porquê?

Tratam-se de endereços privados pois estão na gama 10.0.0.0 - 10.255.255.255.

	Private address range	
Class	start address	finish address
A	10.0.0.0	10.255.255.255
B	172.16.0.0	172.31.255.255
C	192.168.0.0	192.168.255.255

	Public address range	
Class	start address	finish address
A	0.0.0.0	126.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	254.255.255.255

Figura 22: Gama de Endereços e as suas classes

Alínea c

> Porque razão não é atribuído um endereço IP aos switches?

Os *switches*, sendo, neste caso, equipamentos que estão posicionados na camada de nível 2, não acedem à camada de IP (que está no nível 3).

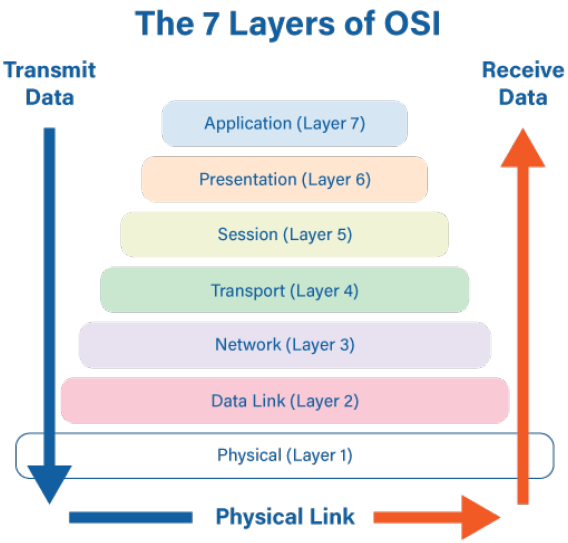


Figura 23: Camadas do modelo OSI

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).

Ao executar o comando IP conseguimos visualizar que todos os pacotes transmitidos foram recebidos, o que é suportado por uma captura de pacotes utilizando o *Wireshark*.

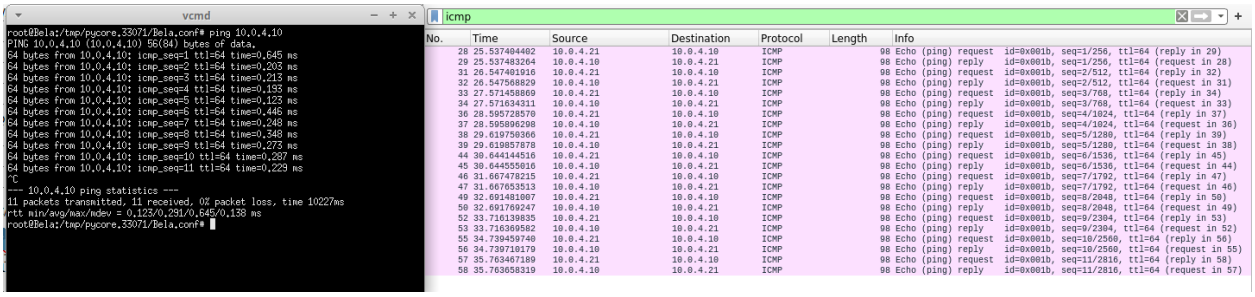


Figura 24: Comando *ping* e captura de pacotes respetiva

Alínea e

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

De modo a verificar a existência de conectividade entre teremos de executar dentro de um departamento, três comandos *ping* (um por cada um dos restantes).

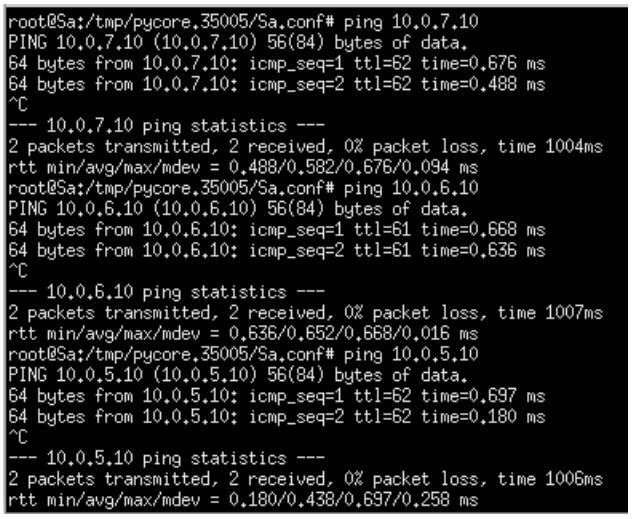


Figura 25: Comandos *ping* e seus resultados

Isto prova a conectividade pois, tendo em conta a topologia criada se *Sa* se consegue conectar com *Sb*, *Sc* e *Sd* então estes também se conseguem conectar entre eles.

Alínea f

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

Tal como conseguimos vizualizar abaixo o portátil Bela consegue aceder ao router de acesso RIPS.

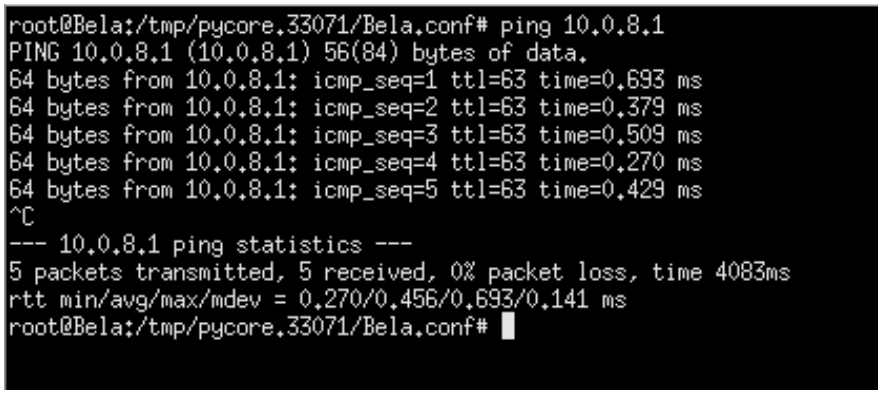


Figura 26: Comando *ping* e seu resultado

Questão 2

> Para o router R_A e o portátil Bela.

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.33071/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.33071/Ra.conf#
```

Figura 27: Output do comando *netstat* no router R_A.

```
root@Bela:/tmp/pycore.33071/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.33071/Bela.conf#
```

Figura 28: Output do comando *netstat* no portátil Bela.

Como podemos ver nos *prints* apresentados, a tabela de encaminhamento do router R_A é muito maior que a do portátil Bela. Se virmos na topologia, o router R_A tem muitas mais interfaces de ligação, enquanto o portátil Bela, apenas tem a ligação com a sub-rede e com o seu departamento. Na primeira tabela, do lado esquerdo, na coluna *Destination*, conseguimos ver todos as redes a que o router R_A está conectado. Na coluna do *Gateway* obtemos respetivamente o ponto de ligação para chegar à rede de destino. A coluna do *Genmask* dá-nos informação acerca da máscara de rede, que neste caso é /24 tendo 8 *bits*

livres para *hosts*. Quanto à coluna das *flags* apenas temos duas opções, onde *U* significa que não passa por nenhum *gateway* (ligação direta) para chegar ao seu destino, enquanto que *UG* significa que não tem uma ligação direta. A coluna *Iface* indica-nos a interface utilizada para chegar ao destino pretendido.

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).

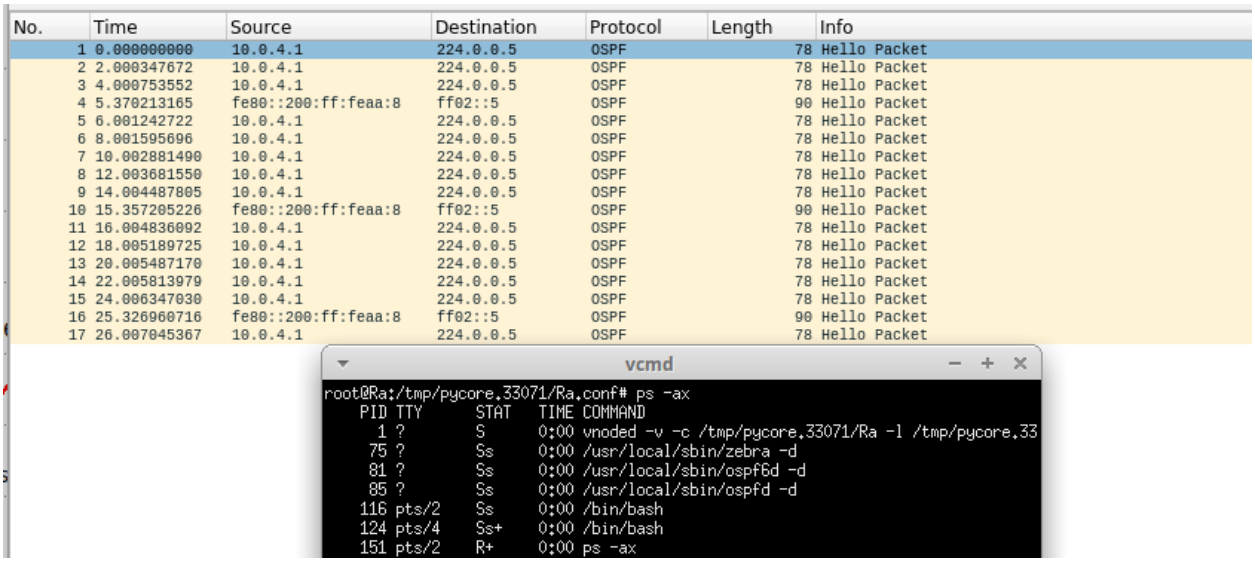


Figura 29: Demonstração do *Wireshark* e do terminal do router R_A

Trata-se de um encaminhamento dinâmico, pois como podemos ver no *print* do *Wireshark* e nos processos com PID 81 e 85 está a correr o protocolo OSPF que verifica a topologia e preenche as tabelas de encaminhamento com os melhores caminhos. De modo a ter um encaminhamento estático teriam de ser adicionadas todas as rotas manualmente.

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.33071/Sa.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@Sa:/tmp/pycore.33071/Sa.conf# route delete 0.0.0.0
SIOCDELRT: No such process
root@Sa:/tmp/pycore.33071/Sa.conf# route delete default
root@Sa:/tmp/pycore.33071/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
```

Figura 30: Comando *Route delete*.

```
root@Jasmine:/tmp/pycore.33071/Jasmine.conf# ping 10.0.4.10
PING 10.0.4.10 (10.0.4.10) 56(84) bytes of data.
^C
--- 10.0.4.10 ping statistics ---
19 packets transmitted, 0 received, 100% packet loss, time 18434ms

root@Jasmine:/tmp/pycore.33071/Jasmine.conf#
```

Figura 31: Teste de *ping* no portátil Jasmine

```
root@Bela:/tmp/pycore.33071/Bela.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=64 time=0.232 ms
64 bytes from 10.0.4.10: icmp_seq=2 ttl=64 time=0.240 ms
64 bytes from 10.0.4.10: icmp_seq=3 ttl=64 time=0.250 ms
64 bytes from 10.0.4.10: icmp_seq=4 ttl=64 time=0.229 ms
^C
--- 10.0.4.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 0.229/0.237/0.250/0.008 ms
root@Bela:/tmp/pycore.33071/Bela.conf#
```

Figura 32: Teste de *ping* no portátil Bela.

Depois de apagarmos a rota pedida, verificamos que os utilizadores da LEI-RC que não pertencem ao departamento (sub-rede) de S_A deixaram de ter acesso ao servidor, mas dentro do departamento os utilizadores continuaram a ter acesso.

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

```
root@Sa:/tmp/pycore.40225/Sa.conf# route add -net 10.0.7.0 netmask 255.255.255>
root@Sa:/tmp/pycore.40225/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
10.0.7.0        10.0.4.1        255.255.255.0   UG        0 0          0 eth0
root@Sa:/tmp/pycore.40225/Sa.conf# route add -net 10.0.8.0 netmask 255.255.255>
root@Sa:/tmp/pycore.40225/Sa.conf# route add -net 10.0.5.0 netmask 255.255.255>
root@Sa:/tmp/pycore.40225/Sa.conf# route add -net 10.0.6.0 netmask 255.255.255>
```

Figura 33: Comandos de adição de rotas.

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@Sa:/tmp/pycore.40225/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
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.40225/Sa.conf#
```

Figura 34: Tabela de encaminhamento.

De modo a garantir que o servidor está novamente acessível utilizamos o comando *ping* para testar conectividade das rotas estátitas entre este (*Sa*) e *hosts* pertencentes às sub-redes adicionadas manualmente.

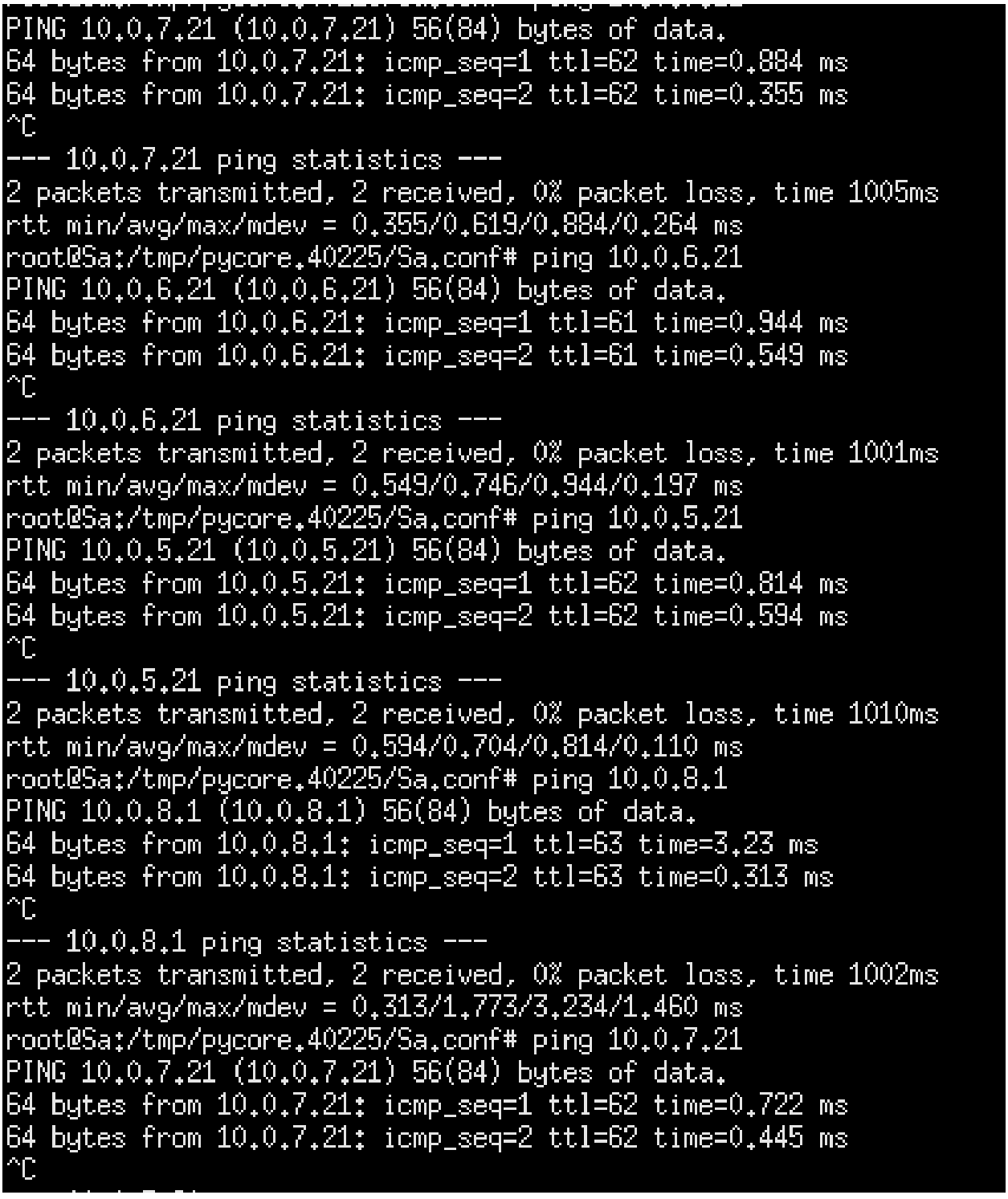


Figura 35: Comandos de *ping*.

Questão 3 - Definição de Sub-redes

> 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. Recorda-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.

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.

O nosso endereço de rede IP é 192.168.34.128/25.
Tendo em conta que atualmente necessitamos de 4 sub-redes (uma para cada departamento) e que a curto prazo o número de departamentos pode vir a aumentar, extendemos a máscara para 28, ou seja, com estes três bits extra ficamos com oito possíveis sub-redes , o que nos deixa com a possibilidade de duplicar o número de departamentos atuais.

- Representação inteira: 192.168.34.128/28
- Representação binária: 11000000.10101000.00100010.10000000

Tendo isto em conta atribuimos a cada um dos routers o menor endereço dentro da sub-rede e incrementamos um valor para cada um dos hosts.

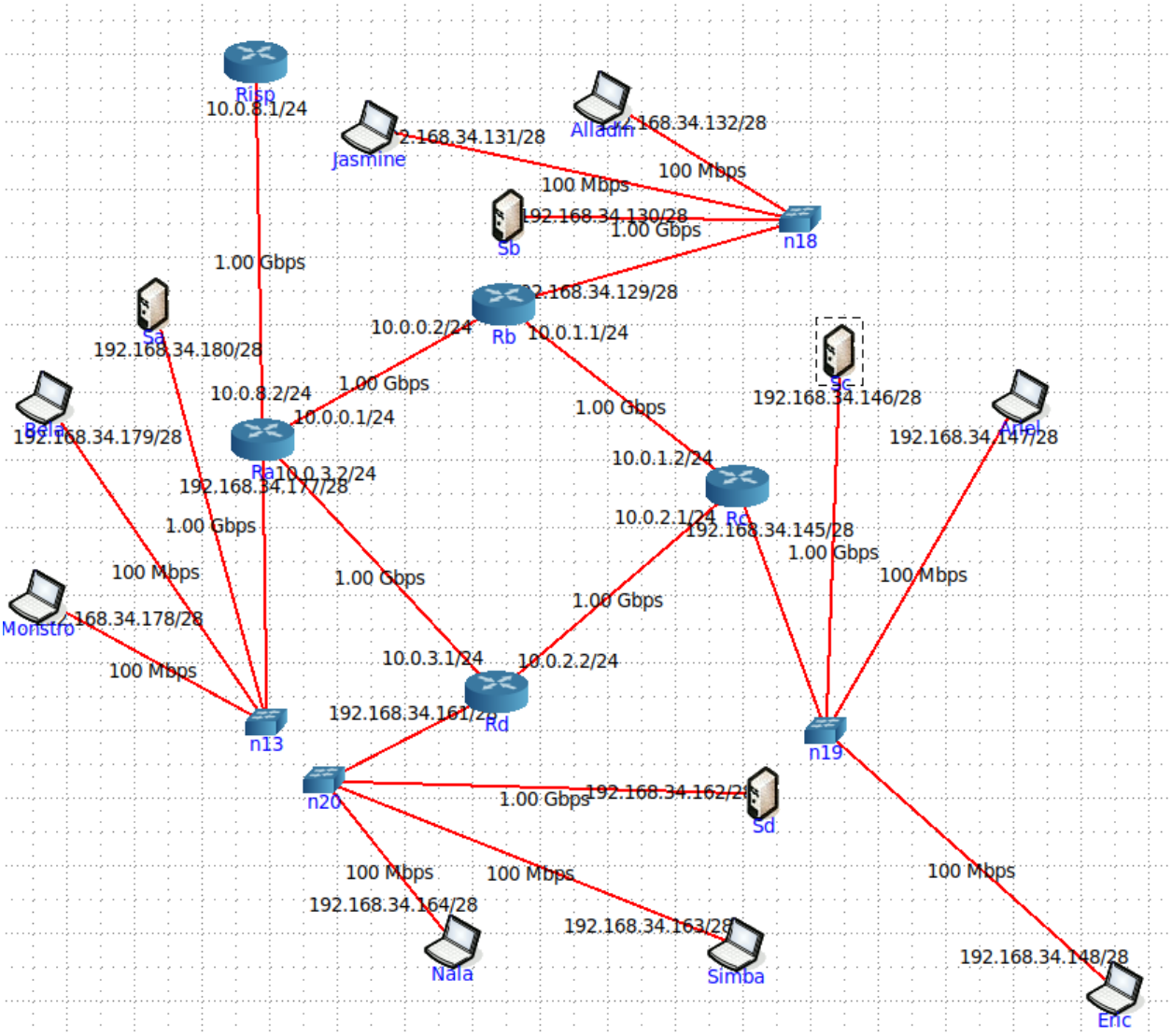


Figura 36: Topologia atualizada

Alínea 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.

Sendo que a máscara de rede é um "/28"a representação inteira respetiva é 255.255.255.240. Como existem 4 bits para os hosts, cada departamento pode interligar $2^4 - 2 = 14$ hosts IP (o .0 e o .15 são reservados).
Tal como constatado anteriormente ficam disponíveis para uso futuro quatro prefixos de sub-

rede, pois existem quatro a serem atualmente usados (um para cada departamento), existem oito possíveis prefixos e "todos os endereços de sub-redes são usáveis". Ou seja, $8 - 4 = 4$.

Prefixos de Rede Utilizados

- SubRede1 - 192.168.34.128/28
- SubRede2 - 192.168.34.144/28
- SubRede3 - 192.168.34.160/28
- SubRede4 - 192.168.34.176/28

Prefixos de Rede Disponíveis

- SubRede5 - 192.168.34.190/28
- SubRede6 - 192.168.34.208/28
- SubRede7 - 192.168.34.224/28
- SubRede8 - 192.168.34.240/28

Alínea 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.

De forma a garantir a conectividade IP interna utilizamos o comando *ping* a partir do servidor de uma das sub-redes com destino a *hosts* das restantes sub-redes e ao *router ISP*.

```
root@Sa:/tmp/pycore.40225/Sa.conf# ping 192.168.34.132
PING 192.168.34.132 (192.168.34.132) 56(84) bytes of data.
64 bytes from 192.168.34.132: icmp_seq=1 ttl=62 time=0.857 ms
64 bytes from 192.168.34.132: icmp_seq=2 ttl=62 time=0.495 ms
64 bytes from 192.168.34.132: icmp_seq=3 ttl=62 time=0.452 ms
^C
--- 192.168.34.132 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2052ms
rtt min/avg/max/mdev = 0.452/0.601/0.857/0.181 ms
root@Sa:/tmp/pycore.40225/Sa.conf# ping 192.168.34.146
PING 192.168.34.146 (192.168.34.146) 56(84) bytes of data.
64 bytes from 192.168.34.146: icmp_seq=1 ttl=61 time=0.876 ms
64 bytes from 192.168.34.146: icmp_seq=2 ttl=61 time=0.645 ms
64 bytes from 192.168.34.146: icmp_seq=3 ttl=61 time=0.796 ms
^C
--- 192.168.34.146 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.645/0.772/0.876/0.095 ms
root@Sa:/tmp/pycore.40225/Sa.conf# ping 192.168.34.162
PING 192.168.34.162 (192.168.34.162) 56(84) bytes of data.
64 bytes from 192.168.34.162: icmp_seq=1 ttl=62 time=0.811 ms
64 bytes from 192.168.34.162: icmp_seq=2 ttl=62 time=0.494 ms
^C
--- 192.168.34.162 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.494/0.652/0.811/0.158 ms
root@Sa:/tmp/pycore.40225/Sa.conf# ping 10.0.8.1
PING 10.0.8.1 (10.0.8.1) 56(84) bytes of data.
64 bytes from 10.0.8.1: icmp_seq=1 ttl=63 time=0.770 ms
64 bytes from 10.0.8.1: icmp_seq=2 ttl=63 time=0.387 ms
^C
--- 10.0.8.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1010ms
rtt min/avg/max/mdev = 0.387/0.578/0.770/0.191 ms
root@Sa:/tmp/pycore.40225/Sa.conf#
```

Figura 37: Comandos *ping* que demonstram a conectividade da topologia atualizada

3. Conclusão

O presente trabalho assentou em duas partes mais gerais, cada uma com o seu grau de dificuldade e complexidade sucessivamente crescente. A primeira parte prendeu-se principalmente com a análise de tráfego através do comando traceroute, explorando conceitos como TTL, RTT, OWD e protocolos ICMP e IP. A última questão já era alusiva à fragmentação IP. A resolução com detalhe da primeira parte permitiu concluir que a análise da topologia em termos de delay se torna muito prática, comprovando a morfologia da topologia através do output do comando traceroute. Para além disso, a utilização do *Wireshark* permitiu estabelecer uma relação entre a topologia, o comando efetuado no terminal e o tráfego capturado. Já na última questão, permitiu analisar a fragmentação com detalhe, sempre comprovando esta análise com conceitos teóricos.

Quanto à segunda parte do trabalho prático, exploramos conceitos de máscara de rede e sub-redes dentro do protocolo IPv4. Abordamos novas técnicas que foram propostas para aumentar a escalabilidade do protocolo IP, mitigar a exaustão dos endereços IPv4 e também reduzir os recursos de memoria necessários nos routers para manter as tabelas de encaminhamento. Na última pergunta, fomos desafiados a alterar os endereços de *IP's* dos *hosts*, personalizando assim a nossa rede consoante o nosso número de grupo. De modo a realizar esta mudança, tivemos de efetuar cálculos de modo a descobrir uma distribuição válida para os endereços.

Concluindo, este trabalho correu bastante bem, conseguimos responder a todas as perguntas, o que nos permitiu aprofundar e consolidar os conceitos teóricos que já aprendemos.