



UNIVERSIDADE DO MINHO

DEPARTAMENTO DE INFORMÁTICA

Redes e Computadores - TP2 - Grupo 56

Filipe Monteiro a80229 Bruno Martins a80410
Márcio Sousa a82400

16 de novembro de 2018

Conteúdo

1	Questões e Respostas	1
1.1	1ª Parte	1
1.2	2ª Parte	11
2	Conclusões	22

1 Questões e Respostas

1.1 1ª Parte

1 - Prepare uma topologia CORE para verificar o comportamento do traceroute. Ligue um host (pc) h1 a um router r2; o router r2 a um router r3, que por sua vez, se liga a um host (servidor) s4. (Note que pode não existir conectividade IP imediata entre h1 e s4 até que o routing estabilize). Ajuste o nome dos equipamentos atribuídos por defeito para a topologia do enunciado.

— a) Ative o wireshark ou o tcpdump no pc h1. Numa shell de h1, execute o comando `traceroute -I` para o endereço IP do *host* s4.

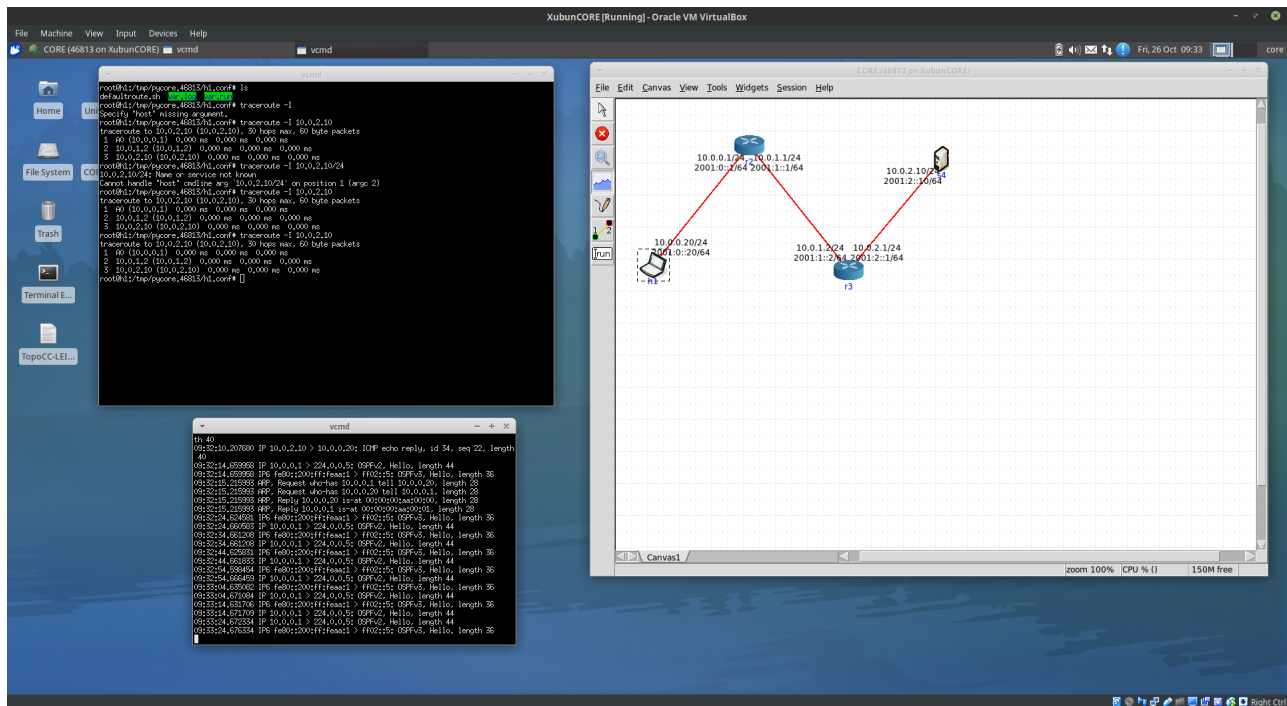


Figura 1: Topologia da rede (à direita), output do comando `traceroute -I` para s4 (em cima) e output do comando `tcpdump` no pc h1 (em baixo).

— b) Registe e analise o tráfego ICMP enviado por h1 e o tráfego ICMP recebido como resposta. Comento os resultados face ao comportamento.

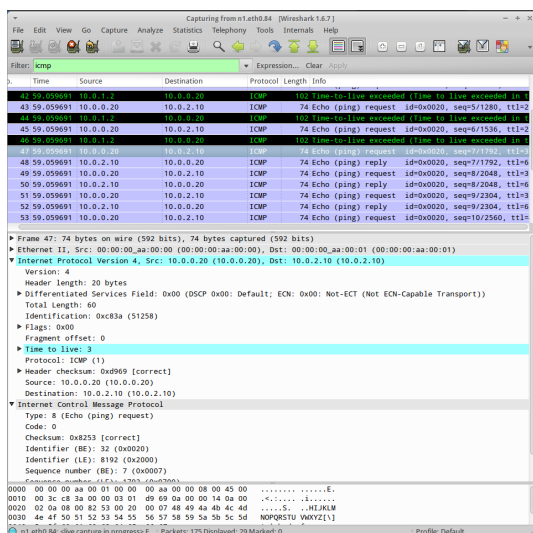
9	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=1/256, ttl=1
10	27.597725	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
11	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=2/512, ttl=1
12	27.597725	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
13	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=3/768, ttl=1
14	27.597725	10.0.0.1	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
15	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=4/1024, ttl=2
16	27.597725	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
17	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=5/1280, ttl=2
18	27.597725	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
19	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=6/1536, ttl=2
20	27.597725	10.0.1.2	10.0.0.20	ICMP	102 Time-to-live exceeded (Time to live exceeded in transit)
21	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=7/1792, ttl=3
22	27.597725	10.0.2.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x002b, seq=7/1792, ttl=62
23	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=8/2048, ttl=3
24	27.597725	10.0.2.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x002b, seq=8/2048, ttl=62
25	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=9/2304, ttl=3
26	27.597725	10.0.2.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x002b, seq=9/2304, ttl=62
27	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=10/2560, ttl=4
28	27.597725	10.0.2.10	10.0.0.20	ICMP	74 Echo (ping) reply id=0x002b, seq=10/2560, ttl=62
29	27.597725	10.0.0.20	10.0.2.10	ICMP	74 Echo (ping) request id=0x002b, seq=11/2816, ttl=4
► Frame 21: 74 bytes on wire (592 bits), 74 bytes captured (592 bits)					
► Ethernet II, Src: 00:00:00_aa:00:00 (00:00:00:aa:00:00), Dst: 00:00:00_aa:00:01 (00:00:00:aa:00:01)					
► Internet Protocol Version 4, Src: 10.0.0.20 (10.0.0.20), Dst: 10.0.2.10 (10.0.2.10)					
► Internet Control Message Protocol					

Figura 2: Primeiras mensagens ICMP enviadas pela nossa máquina.

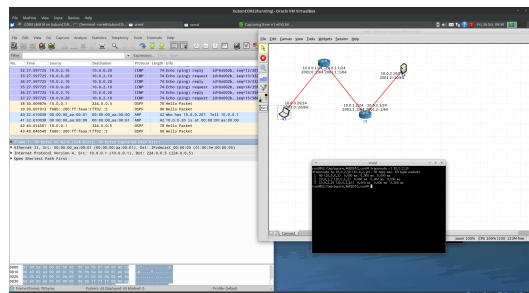
O **traceroute** envia 3 mensagens ICMP com, inicialmente, o valor de TTL=1, e cada router por onde passa o pacote decrementa esta até chegar ao valor 0, onde envia uma resposta com *Time-to-live exceeded*. À medida que vai falhando, envia novas mensagens com TTL superior ao anterior até encontrar um em que o pacote chegue ao destino sem que o TTL fique a 0, recebendo uma resposta *Echo (ping)*. Assim, fica-se a saber quando saltos foram precisos para chegar ao destino (por quantos routers passou).

Como podemos analisar na figura 2, enquanto o TLLT é inferior a 3 a nossa máquina recebe uma resposta *Time-to-live exceeded*, sendo que apenas a partir de TTL igual ou superior a 3 é que recebe uma resposta *Echo (ping)* - o pacote tem de passar por 3 routers antes de chegar ao destino.

— c) Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino s4? Verifique na prtática que a sua resposta está correta.
O valor inicial mínimo do campo TTL para alcançar s4 é de 3.



(a) Wireshark output para os vários TTL's.



(b) Output de *traceroute* com o número de saltos de h1 a s4.

Figura 3: Dois processos diferentes para descobrir o TTL mínimo para chegar a s4 a partir de h1.

Como se vê na figura a, só a partir do $TTL \geq 3$ é que há resposta do servidor. Através da figura b verificamos que, são precisos 3 *hops* para chegar a s4.

— d) Qual o valor médio do tempo de ida-e-volta (Round-Trip Time) obtido?
Apesar de nos ter dado 0 de tempo médio, isto é impossível pois claramente tem de existir uma demora entre o envio e a receção da resposta devido aos saltos e cálculos de routing executados por cada router até ao destino.

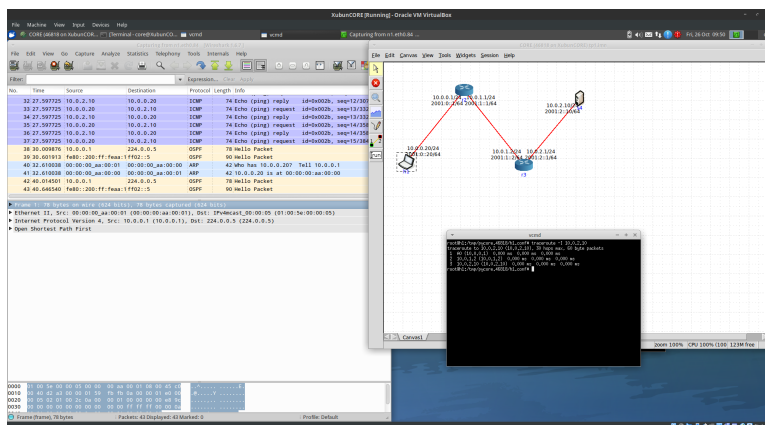


Figura 4: Tempo médio de ida-e-volta entre h1 e s4.

2 - Procedimento a seguir:

Usando o *wireshark* capture o tráfego gerado pelo *traceroute* para os seguintes tamanhos de pacote: (i) sem especificar, i.e., usando o tamanho por defeito; e (ii) 35XX bytes, em que XX é o seu número de grupo. 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 em resposta a esses pedidos.

Selecione a primeira mensagem ICMP capturada (referente a (i) tamanho por defeito) e centre a análise no nível protocolar IP (expanda o *tab* correspondente na janela de detalhe do *wireshark*). Através da análise do cabeçalho IP diga:

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

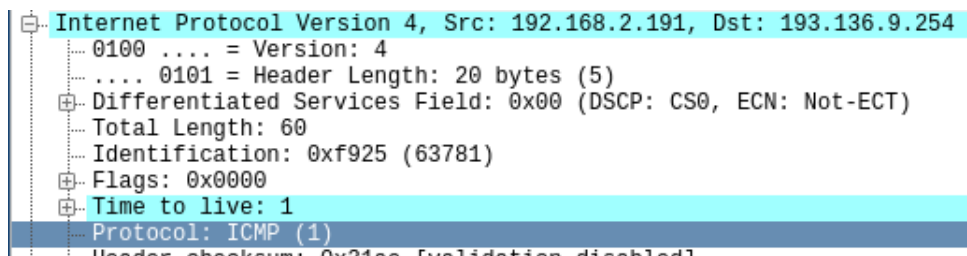
Usando o comando *ifconfig*, verificamos que o nosso IP é de 192.168.2.191.

```
enp3s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.191 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::cdf1:3f82:555c:55ad prefixlen 64 scopeid 0x20<link>
    ether 4c:cc:6a:83:7e:09 txqueuelen 1000 (Ethernet)
    RX packets 13993 bytes 11362918 (11.3 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 11409 bytes 1518766 (1.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
    device interrupt 19
```

Figura 5: Endereço IP da interface Ethernet ligada à rede.

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

Como podemos ver na imagem, no nível do IP tem o campo protocolo com o valor 1 (ICMP).



```
Internet Protocol Version 4, Src: 192.168.2.191, 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: 60
  Identification: 0xf925 (63781)
  Flags: 0x0000
  Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x2100 [validation disabled]
```

Figura 6: Detalhes do datagrama a nível do IP do datagrama em estudo capturado por *wireshark*.

Identifica o protocolo ICMP (*Internet Control Message Protocol*) usado pelo tráfego gerado pelo *traceroute* para ajudar a mapear a rede.

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

Analizando o campo *Header Length* concluímos que o cabeçalho tem 20 *bytes*, o *payload* tem 40

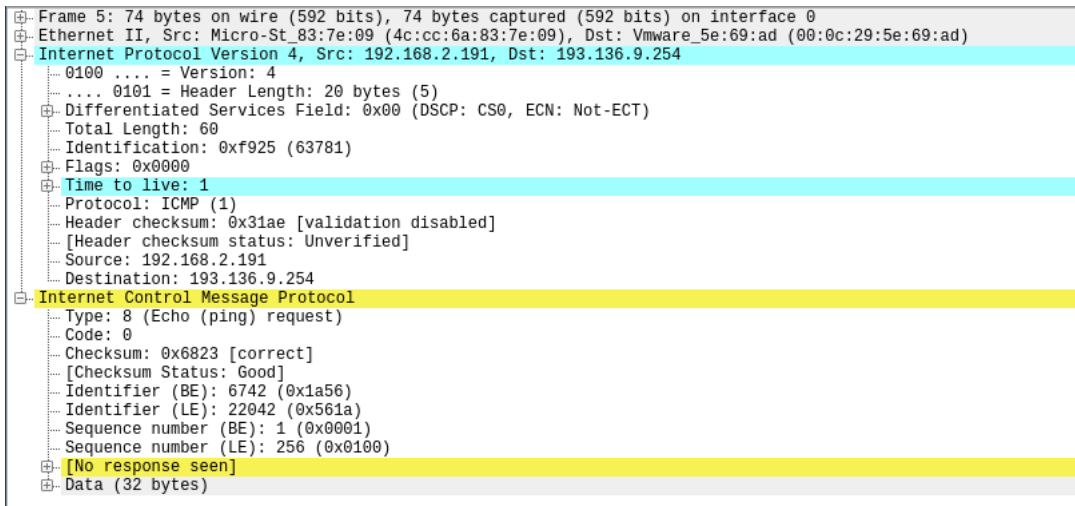


Figura 7: Detalhes do primeiro datagrama ICMP capturado pelo *wireshark*.

bytes e calcula-se retirando ao tamanho total do datagrama o tamanho do cabeçalho: $60 - 20 = 40$ *bytes*. Apesar de parecer 40 *bytes* de payload, na realidade são apenas 32 (como se verifica na figura). Os 8 *bytes* em falta são utilizados no cabeçalho do protocolo UDP usado para transportar as mensagens ICMP.

— d) O datagrama IP foi fragmentado? Justifique.

O datagrama IP não foi fragmentado pois o valor de *Fragment offset* é 0.

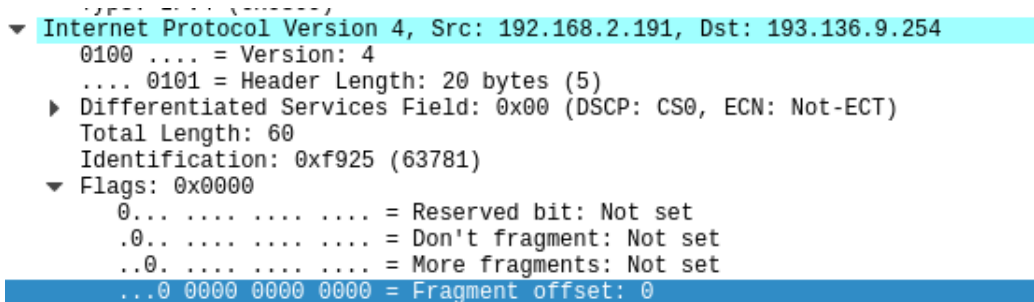


Figura 8: Campos referentes à fragmentação do datagrama em estudo.

— 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, identifique que campos do cabeçalho IP variam de pacote para pacote.

Os campos que variam, analisando as imagens em cima apresentadas, são:

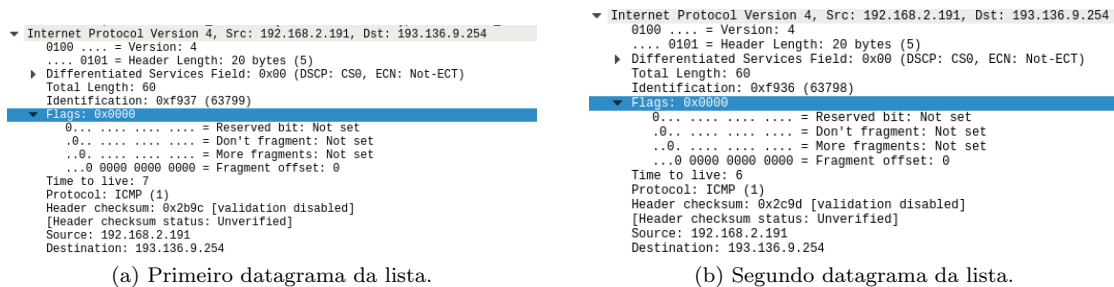


Figura 9: Dois primeiros datagramas capturados.

- *Identification*: campo que identifica o datagrama;
- *Time to live*: campo com o número máximo de saltos que ainda podem ser feitos pelo datagrama. É usado para descobrir por quantos *routers* passa um datagrama para ir de um dispositivo a outro;
- *Header checksum*: Código de verificação do validade do datagrama (controlo de erros);

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

O valor de Identificação é incrementado de 1 em 1 e o TTL é decrementado de 1 em 1 até chegar a 0.

— 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?

O valor do campo TTL é 1 e permanece constante, pois se ocorreu TTL *exceeded* significa que o

```
▼ Internet Control Message Protocol
  Type: 11 (Time-to-live exceeded)
  Code: 0 (Time to live exceeded in transit)
  Checksum: 0xf4ff [correct]
  [Checksum Status: Good]
▼ Internet Protocol Version 4, Src: 192.168.2.191, 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: 60
    Identification: 0xf925 (63781)
  ▶ Flags: 0x0000
  ▶ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0x31ae [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.2.191
    Destination: 193.136.9.254
  ▶ Internet Control Message Protocol
```

Figura 10: Datagrama de uma resposta ICMP com TTL *exceeded*.

pacote não conseguiu chegar ao destino, tendo sido um router o último a decrementar o TTL, indo para 0, descartando este e enviando uma mensagem de erro (Time-to-live exceeded).

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 35XX *bytes*.

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

Houve necessidade de fragmentar o pacote porque o seu tamanho excedia o MTU (*Maximum Transmission Unit*) de 1500 (como nós enviamos um pacote com 3556 *bytes* e o primeiro fragmento tem 1500 de tamanho, deduz-se que o MTU seja de 1500).

— 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? Há mais fragmentos? Qual é o tamanho deste datagrama IP?

```
▼ Internet Protocol Version 4, Src: 192.168.2.191, 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: 0x3879 (14457)
  ▼ Flags: 0x2000, More fragments
    0... .. = Reserved bit: Not set
    .0.. .. = Don't fragment: Not set
    ..1. .... = More fragments: Set
    ...0 0000 0000 0000 = Fragment offset: 0
  ▶ Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0xccba [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.2.191
  Destination: 193.136.9.254
  Reassembled IPv4 in frame: 1803
  ▶ Data (1480 bytes)
```

Figura 11: Detalhes do primeiro fragmento do datagrama em estudo.

O campo *More Fragments* indica que existem mais fragmentos desta mensagem. Como o *Fragment offset* é igual a 0, significa que este é o primeiro fragmento e tem o tamanho de 1500 *bytes*, sendo o tamanho total do datagrama de 3556.

— 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?

Podemos identificar que não se trata do primeiro fragmento através do campo *Fragment offset* e

```
▶ Frame 1802: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0
▶ Ethernet II, Src: Micro-St_83:7e:09 (4c:cc:6a:83:7e:09), Dst: Vmware_5e:69:ad (00:0c:29:5e:69:ad)
▼ Internet Protocol Version 4, Src: 192.168.2.191, 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: 0x3879 (14457)
    ▼ Flags: 0x20b9, More fragments
        0... .. = Reserved bit: Not set
        .0.. .. = Don't fragment: Not set
        ..1. .... = More fragments: Set
        ...0 0000 1011 1001 = Fragment offset: 185
    ▶ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0xcc01 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.2.191
    Destination: 193.136.9.254
    Reassembled IPv4 in frame: 1803
▶ Data (1480 bytes)
```

Figura 12: Detalhes do segundo fragmento do datagrama em estudo.

existem mais fragmentos pois o campo *More fragments* está a 1.

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

Foram criados 3 fragmentos a partir do datagrama original e o último é identificado pelo campo

```
▶ Frame 1803: 610 bytes on wire (4880 bits), 610 bytes captured (4880
▶ Ethernet II, Src: Micro-St_83:7e:09 (4c:cc:6a:83:7e:09), Dst: Vmware
▼ Internet Protocol Version 4, Src: 192.168.2.191, 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: 596
    Identification: 0x3879 (14457)
    ▼ Flags: 0x0172
        0... .. = Reserved bit: Not set
        .0.. .. = Don't fragment: Not set
        ..0. .... = More fragments: Not set
        ...0 0001 0111 0010 = Fragment offset: 370
    ▶ Time to live: 1
    Protocol: ICMP (1)
    Header checksum: 0xfeed [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.2.191
    Destination: 193.136.9.254
```

Figura 13: Detalhes do terceiro (e último) fragmento do datagrama em estudo.

More Fragments a 0 e possuindo um *Fragment offset* maior que o anterior fragmento.

Neste datagrama em particular, podemos verificar que este último fragmento possui um tamanho inferior a todos os outros, podendo então afirmar que é, de facto, o último.

— 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 que mudam no cabeçalho IP são os seguintes (usando as figuras 11, 12 e 13 para comparação):

- *Flags*: o campo *More fragments* fica a 1 até se chegar ao último fragmento e o *Fragment offset* vai aumentando por fragmento;
- *Header checksum*: este muda pois a simples alteração de um campo no cabeçalho origina um código de validação diferente;
- *Total Length*: apenas o último fragmento poderá ter um tamanho inferior aos restantes fragmentos.

A reconstrução é possível através de dois importantes aspectos do cabeçalho: identificação e *Fragment offset*. Com a identificação agregamos os datagramas que possuem este valor igual entre si, ficando assim apenas aqueles fragmentos que pertencem ao datagrama original. Depois, usando o *Fragment offset*, ordenamos os fragmentos por ordem crescente do *offset*, ficando assim com a ordem correta dos fragmentos.

1.2 2ª Parte

Caso de estudo: Considere que a organização MIEI-RC é constituída por três departamentos(A,B e C) e cada departamento possui um *router* de acesso à sua rede local. Estes *routers* de acesso (Ra, Rb e Rc) estão interligados entre si por ligações Ethernet a 1Gbps, formando um anel. Por sua vez, existe um servidor (S1) na rede do departamento C e, pelo menos, três *laptops* por departamento, interligados ao *router* respectivo através de um comutador (*switch*). S1 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 Rext conectado a Rc por uma ligação ponto-a-ponto a 10 Gbps.

Construa uma topologia CORE que reflita a rede local da empresa. Para facilitar a visualização pode ocultar o endereçamento IPV6.

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

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

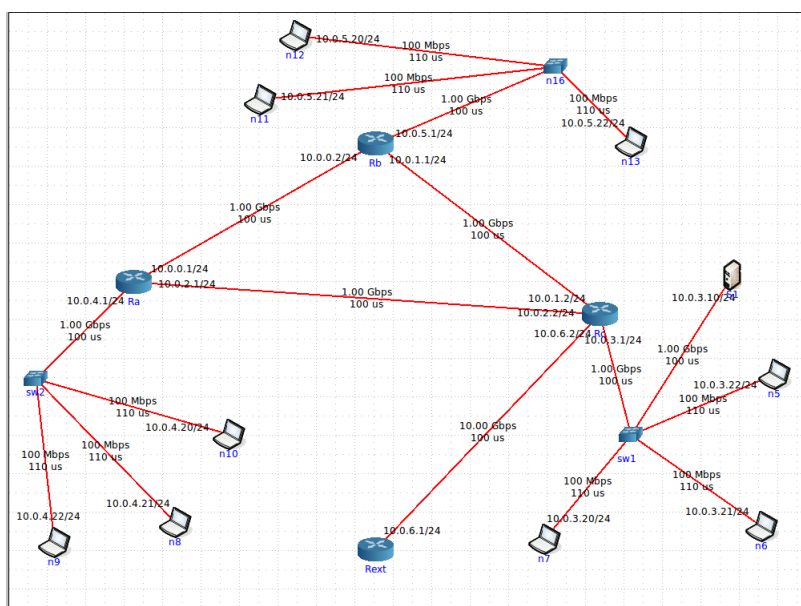


Figura 14: Topologia da rede em estudo.

Cada equipamento possui os IP's visíveis na figura. Existem 7 sub-redes atribuídas (com máscara de 255.255.255.0):

- 10.0.0.0/24
- 10.0.1.0/24
- 10.0.2.0/24
- 10.0.3.0/24
- 10.0.4.0/24
- 10.0.5.0/24
- 10.0.6.0/24

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

Tratam-se de endereços privados pois pertencem à gama 10.0.0.0/8, uma gama standard de redes privadas.

— **c) Porque razão não é atribuído um endereço IP aos *switches*?**

Não são atribuídos IPs aos *switches* pois estes são dispositivos de 2ª camada (*data link layer*), não possuindo IPs (embora possam existir com a 3ª camada (*network layer*) como um router).

— d) Usando o comando ping certifique-se que existe conectividade IP entre os *laptops* dos vários departamentos e o servidor do departamento C (basta certificar-se da conectividade de um *laptop* por departamento).

```
vcmd
root@n9:/tmp/pycore.46823/n9.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=62 time=24.0 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=62 time=24.0 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=62 time=24.0 ms
^C
--- 10.0.3.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 24.001/24.001/24.002/0.178 ms
root@n9:/tmp/pycore.46823/n9.conf#
```

(a) Conectividade entre *laptop* de Ra e servidor S1

```
vcmd
root@n12:/tmp/pycore.46823/n12.conf# ping 10.03.10
PING 10.03.10 (10.3.0.10) 56(84) bytes of data.
From 10.0.5.1 icmp_seq=1 Destination Net Unreachable
From 10.0.5.1 icmp_seq=2 Destination Net Unreachable
From 10.0.5.1 icmp_seq=3 Destination Net Unreachable
From 10.0.5.1 icmp_seq=4 Destination Net Unreachable
^C
--- 10.03.10 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3012ms

root@n12:/tmp/pycore.46823/n12.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=62 time=24.0 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=62 time=24.0 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=62 time=24.0 ms
^C
--- 10.0.3.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 24.001/24.001/24.002/0.126 ms
root@n12:/tmp/pycore.46823/n12.conf#
```

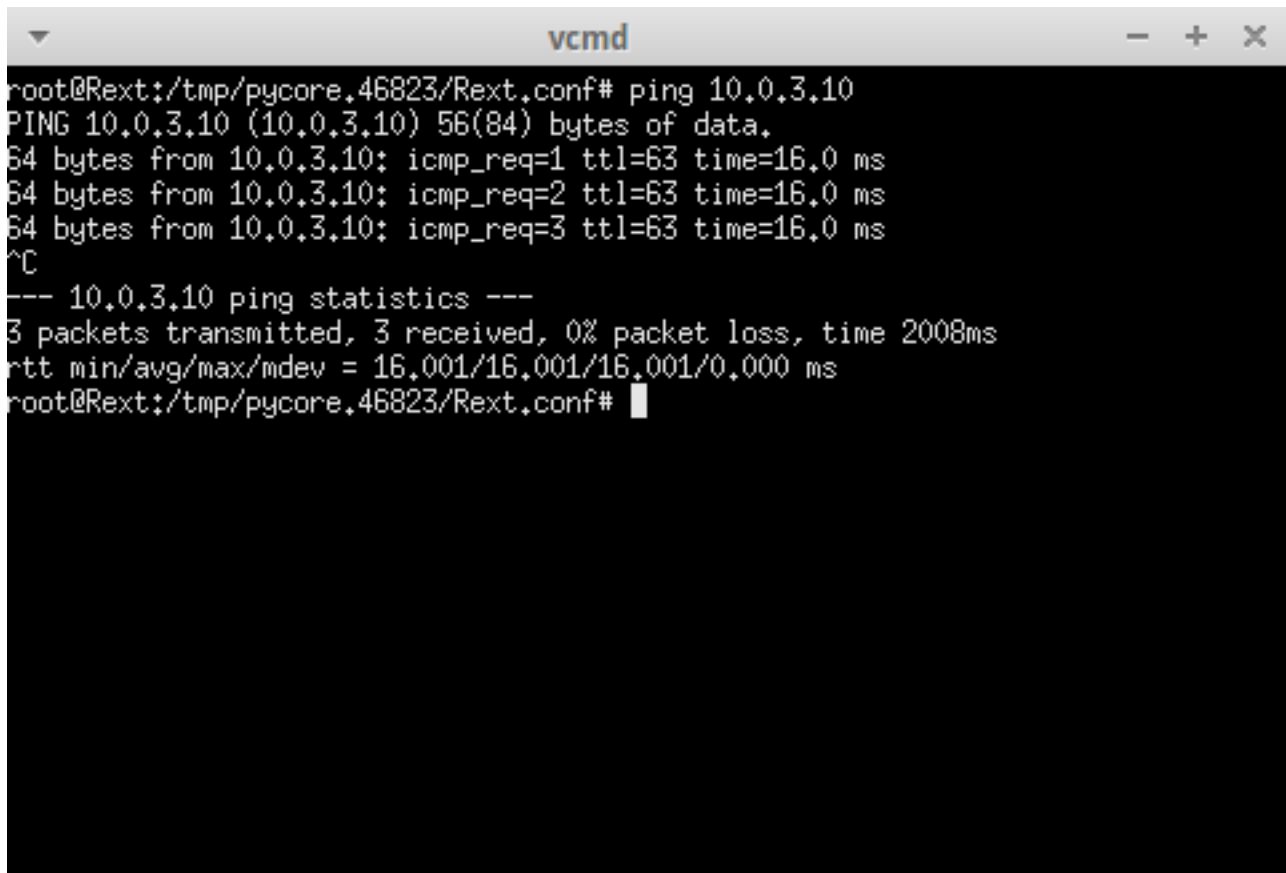
(b) Conectividade entre *laptop* de Rb e servidor S1

```
vcmd
root@n7:/tmp/pycore.46823/n7.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=64 time=16.0 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=64 time=8.00 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=64 time=8.00 ms
^C
--- 10.0.3.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 8.001/10.667/16.001/3.773 ms
root@n7:/tmp/pycore.46823/n7.conf#
```

(c) Conectividade entre *laptop* de Rc e servidor S1

Figura 15: Prova de conectividade entre os *laptops* dos diferentes departamentos e o servidor do departamento C

— e) Verifique se existe conectividade IP do router de acesso Rext para o servidor S1.



```
vcmd
root@Rext:/tmp/pycore.46823/Rext.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data:
64 bytes from 10.0.3.10: icmp_req=1 ttl=63 time=16.0 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=63 time=16.0 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=63 time=16.0 ms
^C
--- 10.0.3.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 16.001/16.001/16.001/0.000 ms
root@Rext:/tmp/pycore.46823/Rext.conf#
```

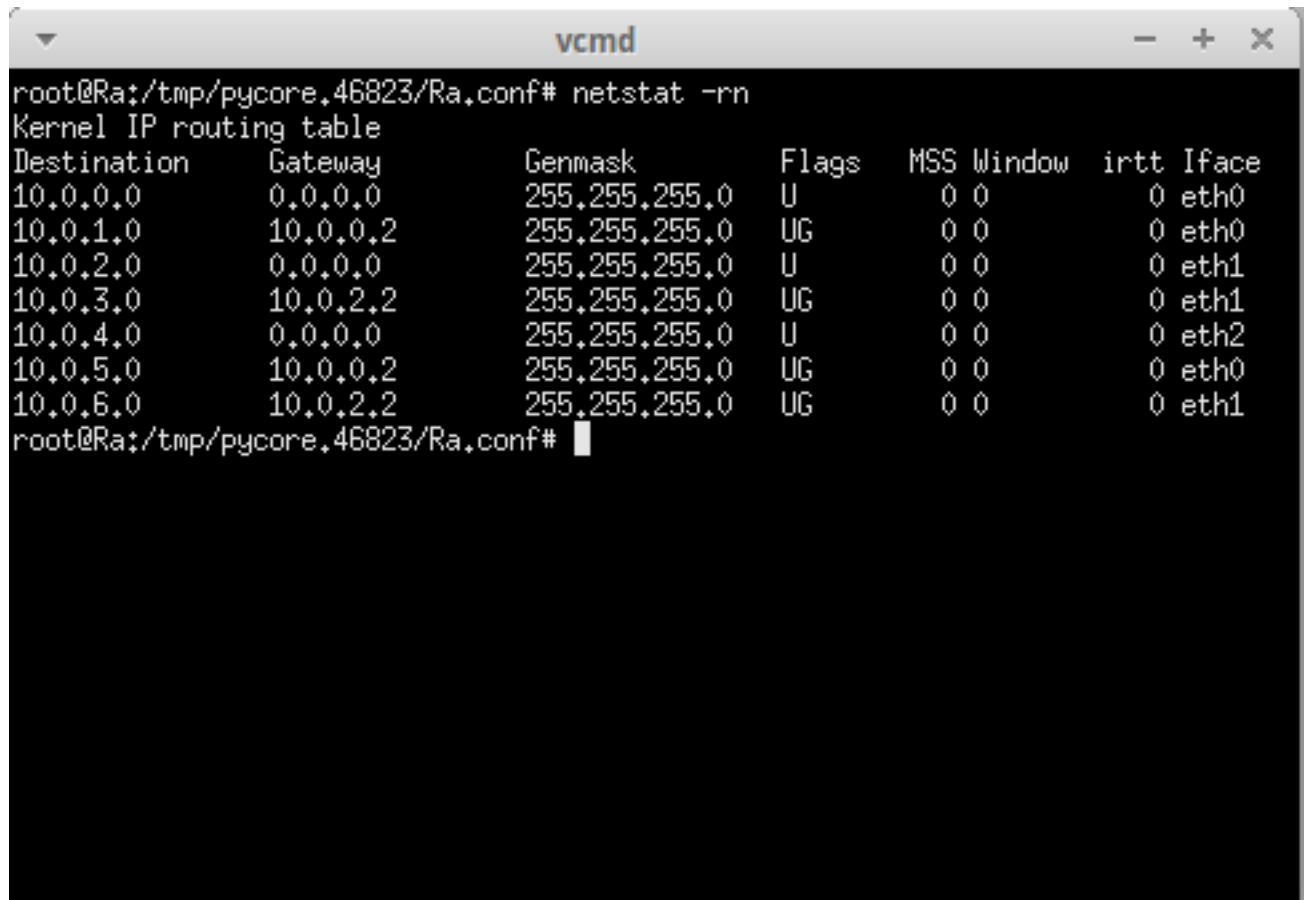
Figura 16: Conectividade IP entre router de acesso Rext e o servidor S1

2 - Para o router e um *laptop* do departamento A:

– 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`).

Extra info:

- Flag U: Informa que a rota está válida.
- Flag G: Informa que *gateway* é um router para depois ser reencaminhado por este (não está ligado diretamente ao IP destino).



```
root@Ra:/tmp/pycore.46823/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         0.0.0.0        255.255.255.0  U       0  0        0 eth1
10.0.3.0         10.0.2.2       255.255.255.0  UG      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         10.0.0.2       255.255.255.0  UG      0  0        0 eth0
10.0.6.0         10.0.2.2       255.255.255.0  UG      0  0        0 eth1
root@Ra:/tmp/pycore.46823/Ra.conf#
```

Figura 17: Tabela de encaminhamento do router Ra

Na tabela do router verificamos as seguintes 7 entradas:

- 10.0.0.0 → 0.0.0.0 : Define que tráfego enviado para um IP destino dentro da sub-rede (10.0.0.0/24) seja encaminhado diretamente para o equipamento sem ter de passar pelo router (estão ligados fisicamente).
- 10.0.1.0 → 10.0.0.2: Esta entrada define que o tráfego enviado para qualquer IP de 10.0.1.0/24 seja enviado para o router em 10.0.0.2 (router Rb) para depois este reencaminhar.

- 10.0.2.0 → 0.0.0.0: O tráfego enviado pra qualquer IP da sub-rede 10.0.2.0/24 é enviado diretamente para o equipamento destino, pois estão ligado fisicamente na mesma sub-rede.
- 10.0.3.0 → 10.0.2.2: Define que o tráfego enviado para qualquer IP de 10.0.3.0/24 seja enviado para o router em 10.0.2.2 (router Rc) para depois este reencaminhar.
- 10.0.4.0 → 0.0.0.0: O tráfego enviado pra qualquer IP da sub-rede 10.0.4.0/24 é enviado diretamente para o equipamento destino, pois estão ligado fisicamente na mesma sub-rede.
- 10.0.5.0 → 10.0.0.2: Esta entrada define que o tráfego enviado para qualquer IP de 10.0.5.0/24 seja enviado para o router em 10.0.0.2 (router Rb) para depois este reencaminhar.
- 10.0.6.0 → 10.0.2.2: O tráfego enviado para qualquer IP de 10.0.6.0/24 seja enviado para o router em 10.0.0.2 (router Rc) para depois este reencaminhar.

```

vcmd
root@n10:/tmp/pycore.46823/n10.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@n10:/tmp/pycore.46823/n10.conf#

```

Figura 18: Tabela de encaminhamento do *laptop* 10 (n10)

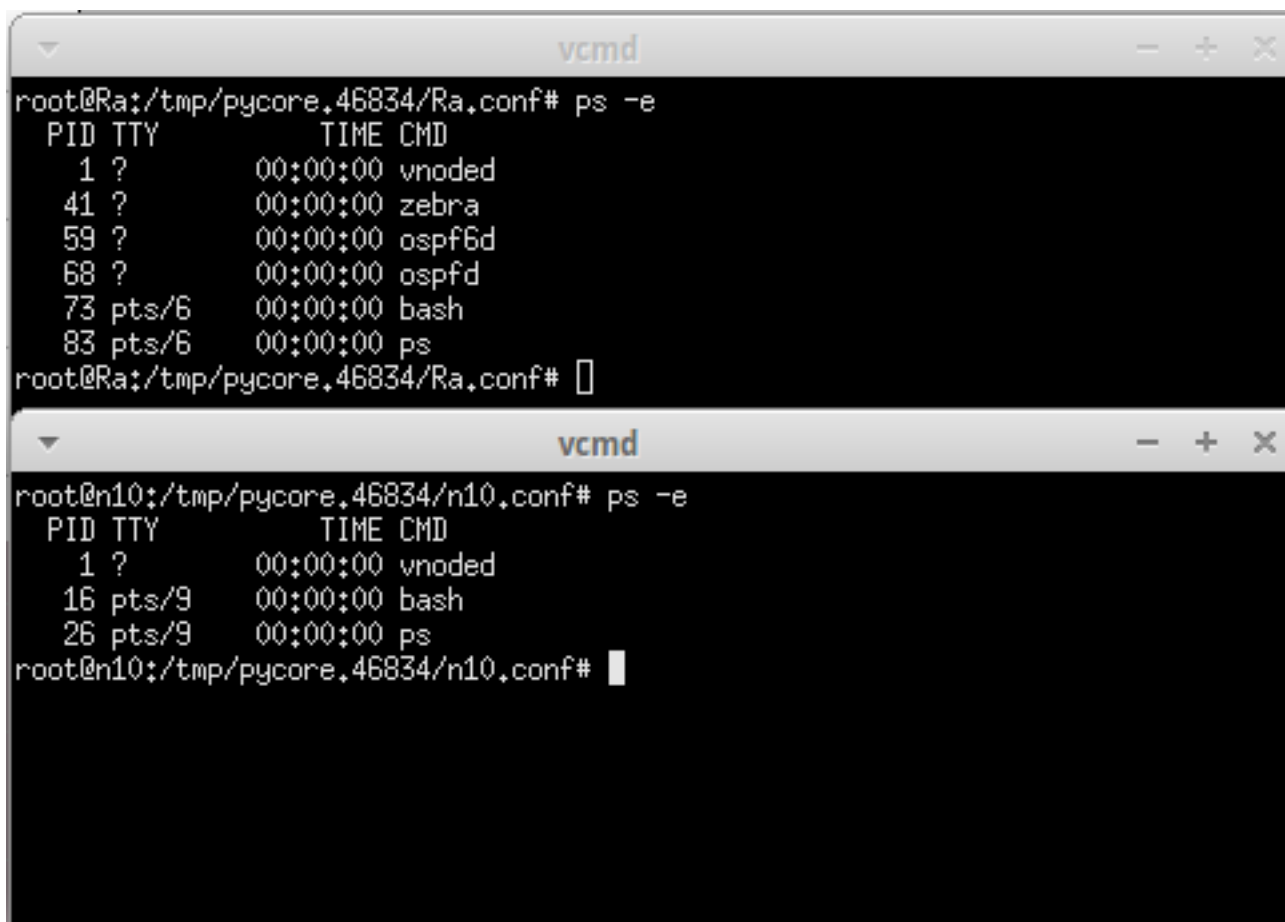
Nesta tabela podemos verificar que tem duas entradas:

- 0.0.0.0 → 10.0.4.1: Esta entrada define que tráfego para qualquer IP de destino seja reencaminhado para o router Ra (10.0.4.1) que depois terá a função de continuar o encaminhamento até ao destino final.
- 10.0.4.0 → 0.0.0.0: Esta entrada define que o tráfego enviado para um IP destino dentro da sub-rede (10.0.4.0/24) seja encaminhado diretamente para o equipamento sem ter de passar pelo router (estão ligados fisicamente).

– b) Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema).

Usando o comando `ps -e` em ambos um router e um *laptop*, verificamos que:

- Router: existe um processo a correr chamdo *ospfd* que é um software the encaminhamento dinâmmico;
- *Laptop*: não existe nenhum processo referente a qualquer tipo de software the encaminhamento dinâmico, logo é estático.



The image shows two terminal windows from a virtual machine environment. The top window is titled 'vcmd' and shows the command prompt 'root@Ra:/tmp/pycore.46834/Ra.conf#' followed by the command 'ps -e'. The output is a table of processes:

PID	TTY	TIME	CMD
1	?	00:00:00	vnoded
41	?	00:00:00	zebra
59	?	00:00:00	ospf6d
68	?	00:00:00	ospfd
73	pts/6	00:00:00	bash
83	pts/6	00:00:00	ps

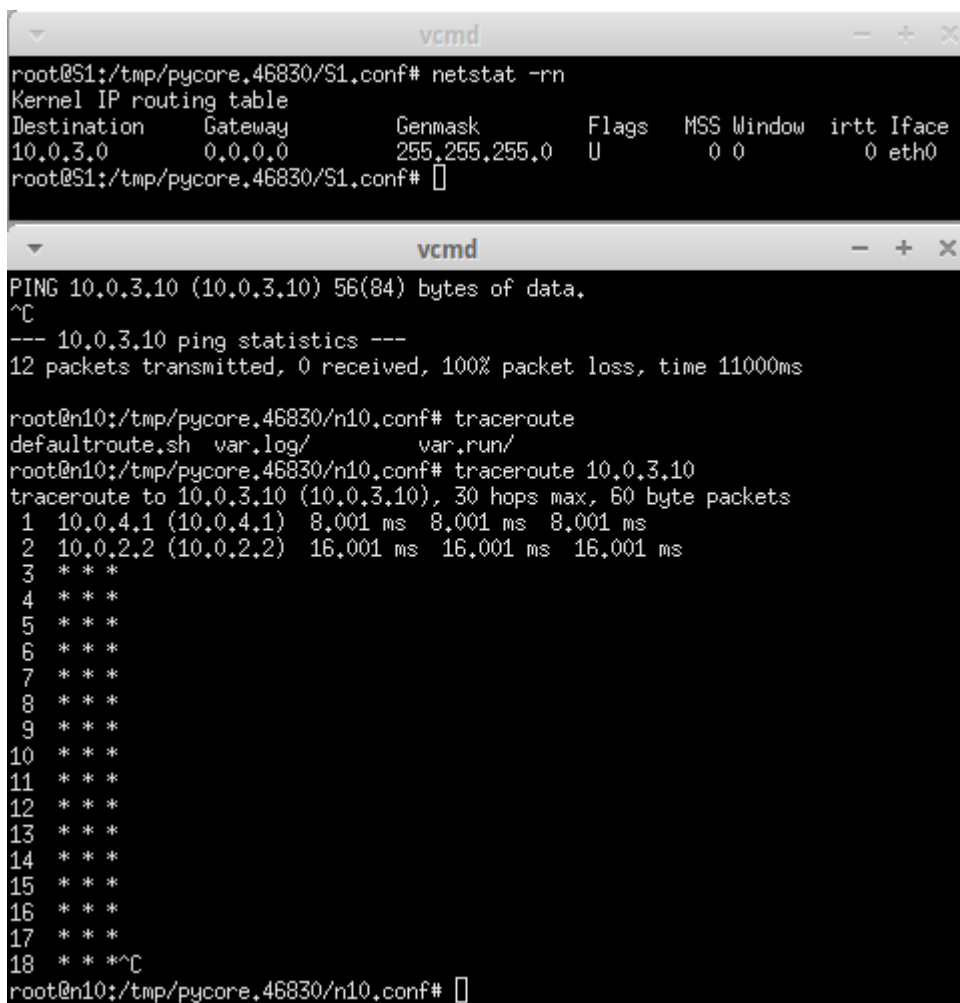
The bottom window is also titled 'vcmd' and shows the command prompt 'root@n10:/tmp/pycore.46834/n10.conf#' followed by the command 'ps -e'. The output is a table of processes:

PID	TTY	TIME	CMD
1	?	00:00:00	vnoded
16	pts/9	00:00:00	bash
26	pts/9	00:00:00	ps

Figura 19: Lista de processos a decorrer no router Ra.

— c) Admita que, por questões administrativas, a rota por defeito (0.0.0.0 ou *default*) deve ser retirada da tabela de encaminhamento do servidor S1 localizado no departamento C. Use o comando `route delete` para o efeito. Que implicações tem esta medida para os utilizadores da empresa que acedem ao servidor? Justifique.

Os utlizadores deixam de poder comunicar com o servidor, pois este não consegue comunicar com redes externas à sua sub-rede - não sabe o que fazer com IP's fora da sua sub-rede). Este apenas consegue comunicar com os dispositivos a que está ligado fisicamente (na mesma sub-rede).



The image shows two terminal windows. The top window, titled 'vcmd', shows the output of the 'netstat -rn' command on server S1. It displays the kernel IP routing table with a single entry for destination 10.0.3.0, gateway 0.0.0.0, and interface eth0. The bottom window, also titled 'vcmd', shows a ping test from a laptop to 10.0.3.10, which fails with 100% packet loss. It also shows a traceroute command that fails at the first hop (10.0.4.1).

```
root@S1:/tmp/pycore.46830/S1.conf# netstat -rn
Kernel IP routing table
Destination      Gateway          Genmask         Flags   MSS Window  irtt Iface
10.0.3.0         0.0.0.0         255.255.255.0   U        0  0        0 eth0
root@S1:/tmp/pycore.46830/S1.conf#

PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
^C
--- 10.0.3.10 ping statistics ---
12 packets transmitted, 0 received, 100% packet loss, time 11000ms

root@n10:/tmp/pycore.46830/n10.conf# traceroute
defaultroute.sh  var.log/          var.run/
root@n10:/tmp/pycore.46830/n10.conf# traceroute 10.0.3.10
traceroute to 10.0.3.10 (10.0.3.10), 30 hops max, 60 byte packets
 1  10.0.4.1 (10.0.4.1)  8.001 ms  8.001 ms  8.001 ms
 2  10.0.2.2 (10.0.2.2) 16.001 ms 16.001 ms 16.001 ms
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *^C
root@n10:/tmp/pycore.46830/n10.conf#
```

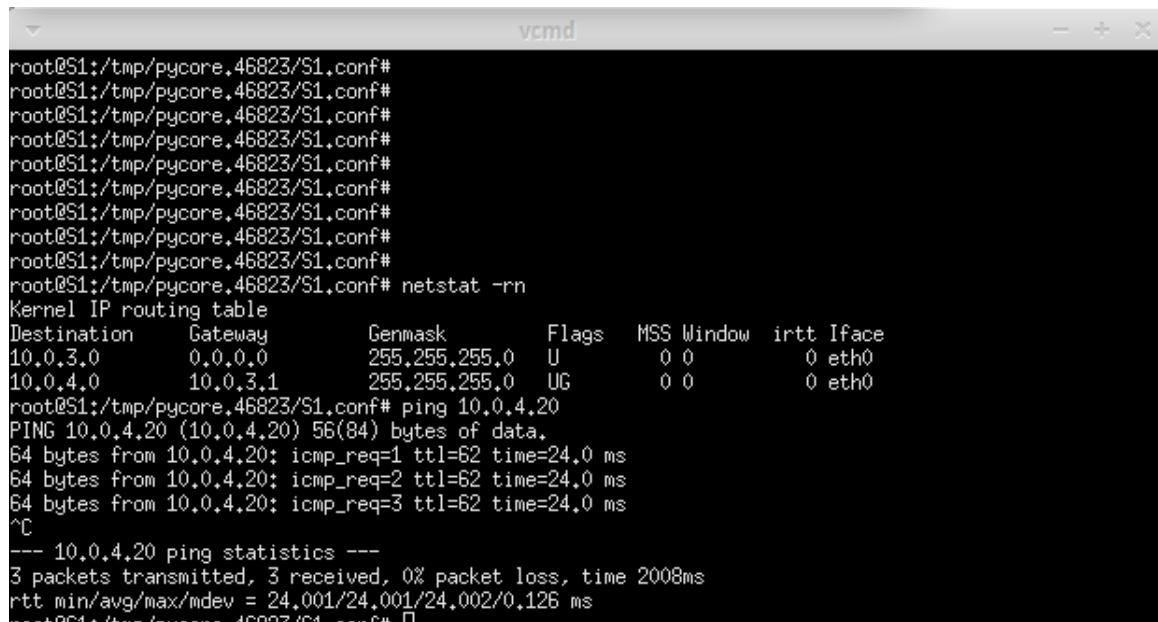
Figura 20: Servidor S1 não acessível por um *laptop* fora da sua sub-rede.

— d) Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor S1, por forma a contornar a restrição imposta na alínea c). Utilize para o efeito o comando `route add` e registe os comandos que usou.

Comandos usados:

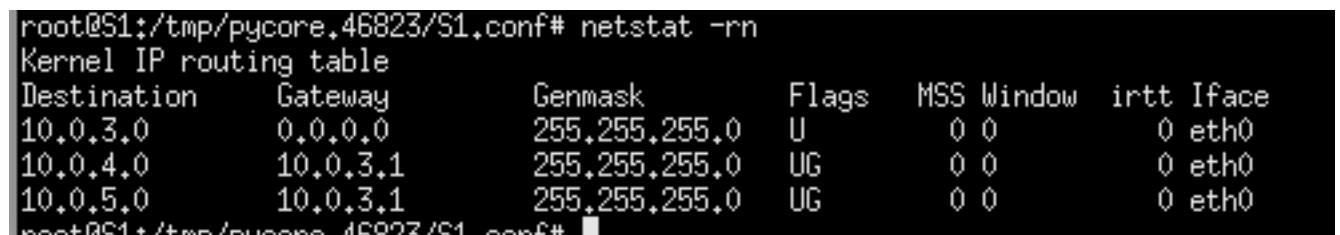
- `route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.3.1`
- `route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.3.1`

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



```
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf#  
root@S1:/tmp/pycore.46823/S1.conf# netstat -rn  
Kernel IP routing table  
Destination      Gateway          Genmask          Flags   MSS Window  irtt Iface  
10.0.3.0          0.0.0.0          255.255.255.0    U        0 0        0 eth0  
10.0.4.0          10.0.3.1          255.255.255.0    UG       0 0        0 eth0  
root@S1:/tmp/pycore.46823/S1.conf# ping 10.0.4.20  
PING 10.0.4.20 (10.0.4.20) 56(84) bytes of data:  
64 bytes from 10.0.4.20: icmp_req=1 ttl=62 time=24.0 ms  
64 bytes from 10.0.4.20: icmp_req=2 ttl=62 time=24.0 ms  
64 bytes from 10.0.4.20: icmp_req=3 ttl=62 time=24.0 ms  
^C  
--- 10.0.4.20 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2008ms  
rtt min/avg/max/mdev = 24.001/24.001/24.002/0.126 ms  
root@S1:/tmp/pycore.46823/S1.conf#
```

Figura 21: Prova de conectividade entre um computador da sub-rede 10.0.4.0/24 (especificamente do equipamento 10.0.4.20) e o servidor S1, com a nova rota definida.



```
root@S1:/tmp/pycore.46823/S1.conf# netstat -rn  
Kernel IP routing table  
Destination      Gateway          Genmask          Flags   MSS Window  irtt Iface  
10.0.3.0          0.0.0.0          255.255.255.0    U        0 0        0 eth0  
10.0.4.0          10.0.3.1          255.255.255.0    UG       0 0        0 eth0  
10.0.5.0          10.0.3.1          255.255.255.0    UG       0 0        0 eth0  
root@S1:/tmp/pycore.46823/S1.conf#
```

Figura 22: Nova Tabela de encaminhamento do servidor S1.

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

Para definir endereços de sub-rede é necessário usar a parte prevista para endereçamento de *host*, não sendo possível alterar o endereço de rede original. Recorde-se que o *sub-redeting*, 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* se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.

— a) Considere que dispõe apenas do endereço de rede IP 172.XX.48.0/20, em que XX é o decimal correspondendo ao seu número de grupo (PLXX). Defina um novo esquema de endereçamento para as redes dos departamentos (mantendo a rede de acesso e core inalteradas) e atribua endereços às interfaces dos vários sistemas envolvidos. Deve justificar as opções usadas.

Como são apenas 3 departamentos, para possibilitar a expansão destes e melhor organização da rede destes, definimos 3bits para sub-rede, tornando-se a máscara de rede /23. Sendo que dispomos do endereço IP 172.56.48.0/20, criamos o seguinte esquema:

SA	172.56.48.0/23	172.56.49.0/23	510 hosts	Departamento A
SB	172.56.50.0/23	172.56.51.0/23	510 hosts	Departamento A
SC	172.56.52.0/23	172.56.53.0/23	510 hosts	Departamento B
SD	172.56.54.0/23	172.56.55.0/23	510 hosts	Departamento B
SE	172.56.56.0/23	172.56.57.0/23	510 hosts	Departamento C
SF	172.56.58.0/23	172.56.59.0/23	510 hosts	Departamento C
SG	172.56.60.0/23	172.56.61.0/23	510 hosts	Outros
SH	172.56.62.0/23	172.56.63.0/23	510 hosts	Outros

Nota: não retiramos a primeira e última sub-redes, pois apesar de estas terem sido reservadas no passado, atualmente os routers modernos têm a capacidade de os usarem.

Assim, o departamento A ficará com SA e, caso queira expandir e para facilitar, com a gama de SB, o departamento B com SC e SD e o departamento C com SE e SF.

Mas é importante referir que a organização da rede depende muito do cenário em que será aplicado, sendo este um cenário imaginado por nós onde se aplique esta determinada organização.

— b) Qual a máscara de rede que usou (em formato decimal)? Quantos *hosts* IP pode interligar em cada departamento? Justifique.

A máscara de rede usada foi /23, ou seja, 255.255.254.0, criando 8 sub-redes, podendo cada departamento pode interligar 510 *hosts*. Optamos por esta escolha, visto que, com 8 sub-redes permite a cada departamento (três por agora) usar até duas gamas de IPs, sobrando ainda outras duas para uma possível adição de uma outra sub-rede. Assim, cada departamento pode usar um máximo de 1020 *hosts* caso use as suas duas sub-redes respectivas.

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

A cada departamento atribuímos IP's da gama da sua sub-rede e para testar conectividade usamos o comando **ping** com o IP de máquinas de outras sub-redes.

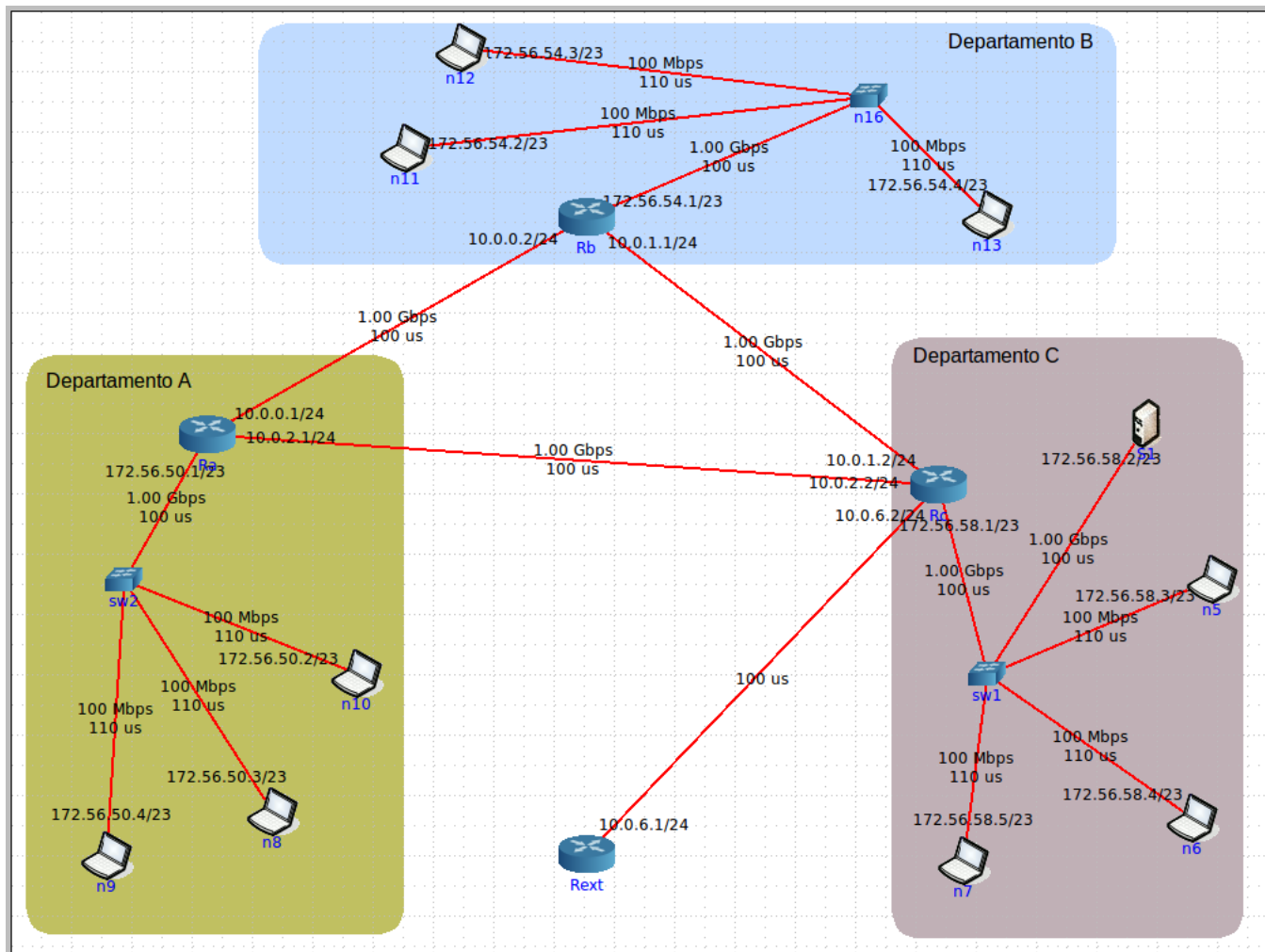
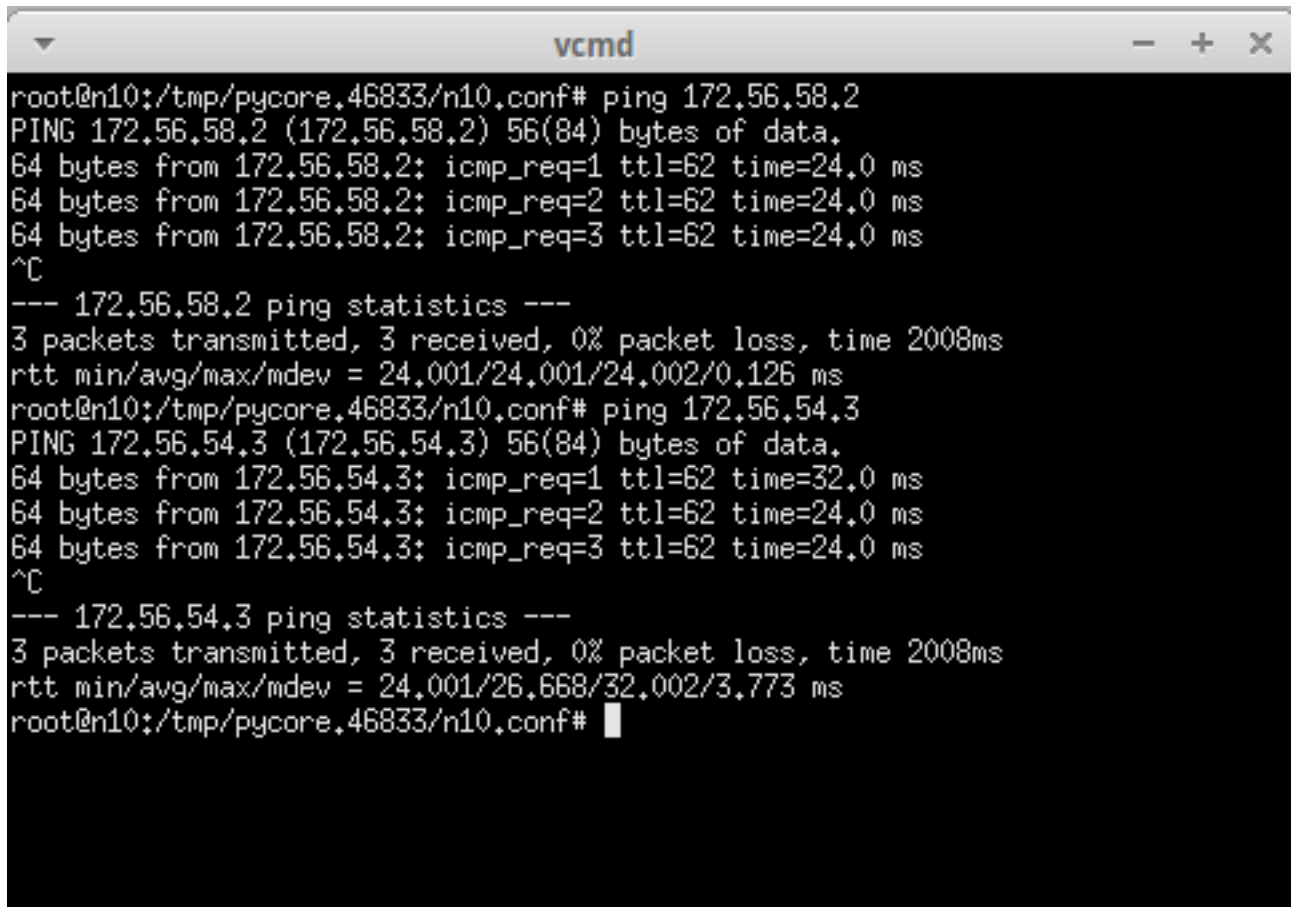


Figura 23: Nova Topologia da rede em estudo.



```
vcmd
root@n10:/tmp/pycore.46833/n10.conf# ping 172.56.58.2
PING 172.56.58.2 (172.56.58.2) 56(84) bytes of data.
64 bytes from 172.56.58.2: icmp_req=1 ttl=62 time=24.0 ms
64 bytes from 172.56.58.2: icmp_req=2 ttl=62 time=24.0 ms
64 bytes from 172.56.58.2: icmp_req=3 ttl=62 time=24.0 ms
^C
--- 172.56.58.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 24.001/24.001/24.002/0.126 ms
root@n10:/tmp/pycore.46833/n10.conf# ping 172.56.54.3
PING 172.56.54.3 (172.56.54.3) 56(84) bytes of data.
64 bytes from 172.56.54.3: icmp_req=1 ttl=62 time=32.0 ms
64 bytes from 172.56.54.3: icmp_req=2 ttl=62 time=24.0 ms
64 bytes from 172.56.54.3: icmp_req=3 ttl=62 time=24.0 ms
^C
--- 172.56.54.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2008ms
rtt min/avg/max/mdev = 24.001/26.668/32.002/3.773 ms
root@n10:/tmp/pycore.46833/n10.conf#
```

Figura 24: Prova de conectividade entre as diferentes sub-redes: n10 (Dep A) e S1 (Dep C); n10 (Dep A) e n11 (Dep B).

2 Conclusões

Com a resolução deste trabalho foi possível abordar as temáticas da Unidade Curricular de Redes e Computadores de forma mais aprofundada, consolidando o conhecimento adquirido nas aulas teóricas. A primeira secção teve como foco principal o protocolo IPv4 com especial atenção nos datagramas IP e na fragmentação incluídos. Através da utilização da tecnologia *Core* foi possível representar topologias de redes e desta forma analisar como os pacotes se comportam nas redes. Numa primeira abordagem foi analisado o TTL e do quão importante é ter um TTL baixo para que os pacotes cheguem rapidamente ao seu destino. Para análise do TTL foi necessária a ferramenta *Wireshark* que mostra aos utilizadores os pacotes que estão em movimento na rede. Ainda nesta secção foi explorado como o protocolo ICMP é incluído no pacote (através da utilização de um header), o tamanho que este ocupa bem como o que verdadeiramente é a informação que se pretende transmitir. Podemos também analisar e entender como se processa a fragmentação de um pacote, que ocorre quando não é possível enviar a informação todo num pacote e se tem de enviar em vários, e de como os pacotes se identificavam entre si.

A segunda secção, embora ainda dedicada ao protocolo IPv4, teve como temáticas o endereçamento e encaminhamento IP. Para isso foi utilizado novamente o *Core* para representar uma topologia de vários departamentos. Foi desta forma analisada se os IP's da topologia eram privados ou públicos e a razão pela qual diferentes computadores conseguem comunicar entre diferentes sub-redes utilizando routers. Quanto ao endereçamento foi verificado se o encaminhamento era estático ou dinâmico podendo assim assimilar os conceitos que aparentam ser mais teóricos de uma forma mais aproximada à realidade, concluindo que nos dias de hoje, nas tecnologias usadas os routers são bastante inteligentes e fazem um encaminhamento dinâmico para máxima eficiência.

Foi também percebido a importância de uma boa organização da rede, para melhor utilização do espaço de endereçamento associado a esta - tudo dependendo da utilização final e futuras expansões desta mesma. A criação de sub-redes, a atribuição destas a diferentes zonas da rede e o processo de encaminhamento, foram todos importantes aspetos estudados e bem assimilados.