

TP2: Protocolo IP

Diogo Afonso Costa, Daniel Maia, and Vitor Castro

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a78034,a77531,a77870}@alunos.uminho.pt

1 Introdução

O principal objectivo deste trabalho é o estudo do Internet Protocol (IP) nas suas principais vertentes (estudo do formato de um pacote ou datagrama IP, endereçamento IP, encaminhamento IP e fragmentação de pacotes IP), bem como o estudo do protocolo IPv4 com enfâse no endereçamento e encaminhamento IP. Para tal, foram usadas ferramentas como o *CORE*, *Wireshark*, *Traceroute*, *Netstat*, entre outras.

2 Parte I - Datagramas e Fragmentação

2.1 Exercício 1.b.

Questão

Registe e analise o tráfego ICMP enviado por n1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

Resposta

Inicialmente, é feito o envio de um pacote com o TTL = 1, que não recebe resposta porque *time exceeded message*, que acontece quando um datagrama chega a um *gateway* com TTL = 0 (será neste ponto que será enviada uma resposta para a fonte do pacote, notificando da situação). Então, o TTL é progressivamente incrementado até ser igual a 3, sendo que aí já recebe mensagem de resposta. É também possível verificar que há envio de vários pacotes com o TTL = 1 e TTL = 2, prevenindo a perda de um pacote ou uma falha no sistema, atuando assim como uma redundância. Depois vemos que a partir que o TTL = 3 o TTL é incrementado a TTL= 8 e as replies são feitas com TTL = 62 constantemente, por definição.

O tamanho do pacote é 74 bytes mas a mensagem tem 60 bytes de tamanho sendo que 14 bytes são para o header do nível 2. Nesta mensagem de 60 bytes estão incluídos 20 bytes de header length (variáveis de acordo com o protocolo IPv4) e nos 40 bytes restantes temos o ICMP header (8 bytes) e ICMP payload (32 bytes), sendo Data resultante de 32 bytes. Vemos que a mensagem de 32 bytes aumentou em 42 bytes até o nível 2 de rede.

Realização

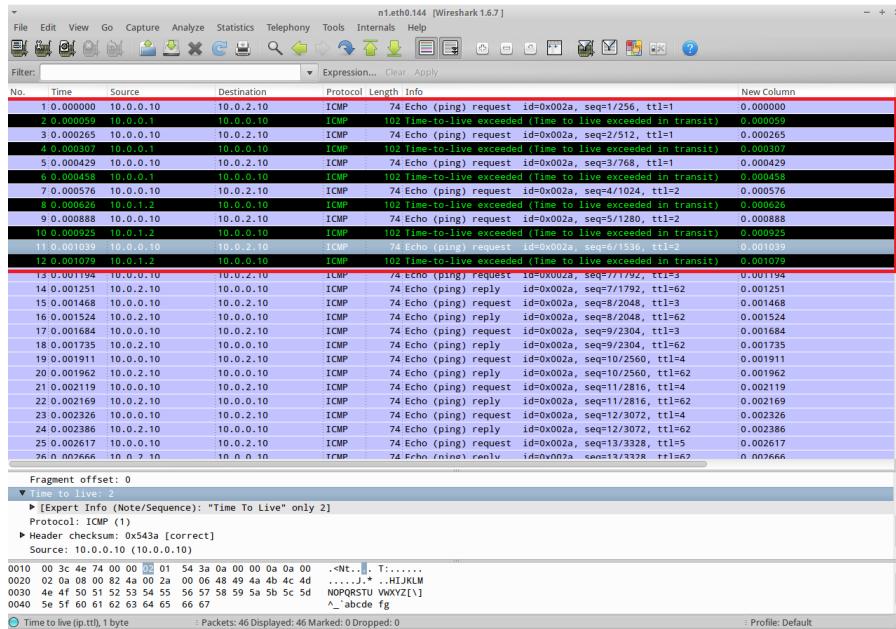


Fig. 1: TTL excedido.

No.	Time	Source	Destination	Protocol	Length	Info	New Column
38	0.011434	10.0.0.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=19/4864, ttl=62	0.011434
39	0.016216	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=20/5120, ttl=7	0.016216
40	0.016277	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=20/5120, ttl=62	0.016277
41	0.017445	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=21/5376, ttl=7	0.017445
42	0.017498	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=21/5376, ttl=62	0.017498

Frame 41: 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: 10.0.0.10 (10.0.0.10), Dst: 10.0.2.10 (10.0.2.10)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))
Total Length: 60
Identification: 0x178d (6029)
Flags: 0x00
Fragment offset: 0
Time to live: 7
Protocol: ICMP ()
Header checksum: 0x8621 [correct]
Source: 10.0.0.10 (10.0.0.10)
Destination: 10.0.2.10 (10.0.2.10)
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0xb23b [correct]
Identifier (BE): 42 (0x002a)
Identifier (LE): 10752 (0x2a00)
Sequence number (BE): 21 (0x0015)
Sequence number (LE): 5376 (0x1500)
(Response In: 42)
Data (32 bytes)

Fig. 2: Tamanho do pacote e divisão.

2.2 Exercício 1.c.

Questão

Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino n4? Verifique na prática que a sua resposta está correta.

Resposta

O TTL mínimo necessário para alcançar o destino n4 é 3. Isto é confirmado pelo *wire-shark*, que mostra que as tentativas com TTL menor que 3 resultou em *time exceeded message*.

Realização

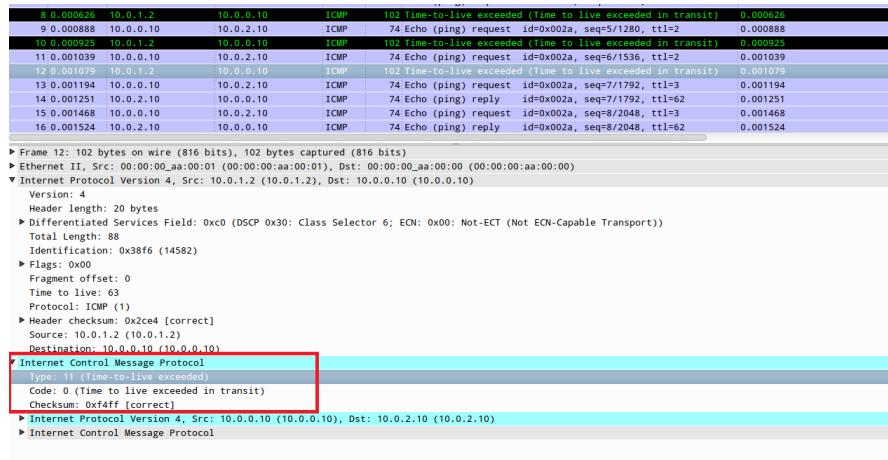


Fig. 3: Mensagem com ICMP retornando TTL excedido.

2.3 Exercício 1.d.

Questão

Qual o valor médio do tempo de ida-e-volta (Round-Trip Time) obtido?

Resposta

O Round-Trip Time é igual à soma do tempo de envio com o tempo de resposta. Logo, fazendo a soma das médias dos tempos, temos um resultado de Round-Trip Time médio de 0.00015 s (tempo de ida/request) + 0.00005 s (tempo de volta/reply) = 0.00020 s. É de notar que tanto o tempo de request como o de reply se mantêm relativamente constantes. Também se pode analisar este tempo recorrendo ao tempo relativo entre duas mensagens da mesma source.

Realização

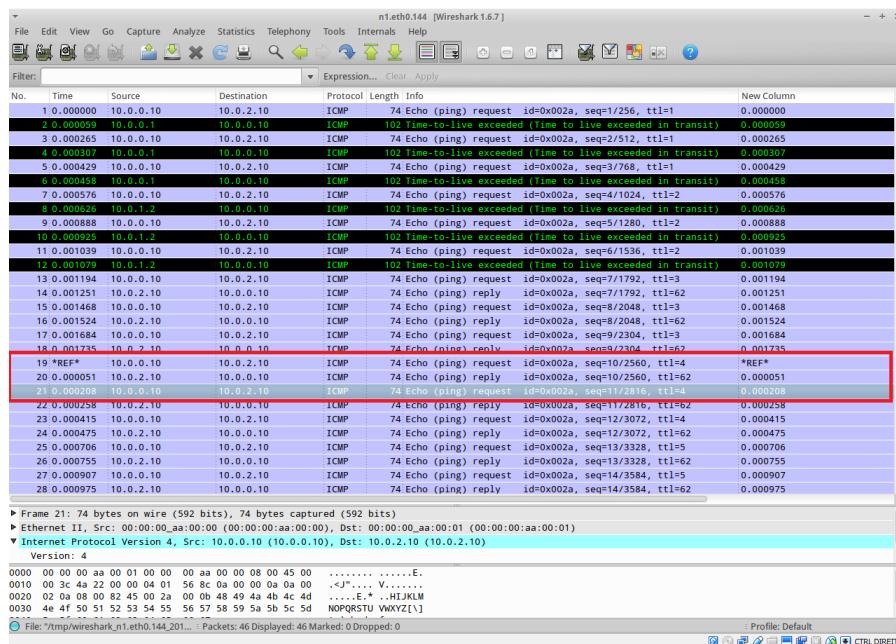


Fig. 4: Round-Trip Time.

2.4 Exercício 2.a.

Questão

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

Resposta

O endereço IP é 192.168.100.216.

Realização

43	3.903910826	192.168.100.216	192.168.100.216	UNDEF	519	Stduard query response 0x7472 A mdcu.unicom.br A 192.168.9.240 NS DNS
44	3.903910826	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=1/256, ttl=1 (no response found!)
45	3.903938117	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=2/512, ttl=1 (no response found!)
46	3.903947143	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=3/768, ttl=1 (no response found!)
47	3.903956246	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=4/1024, ttl=2 (no response found!)
48	3.903964672	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=5/1280, ttl=2 (no response found!)
49	3.903970621	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=6/1536, ttl=2 (no response found!)
50	3.903975159	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=7/1792, ttl=3 (reply in 63)
51	3.903982289	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=8/2048, ttl=3 (reply in 64)
52	3.903988913	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=9/2304, ttl=3 (reply in 66)
53	3.903994876	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=10/2560, ttl=4 (reply in 67)
54	3.904000785	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=11/2816, ttl=4 (reply in 68)
55	3.904009311	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=12/3072, ttl=4 (reply in 69)
56	3.904015173	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=13/3328, ttl=5 (reply in 70)
57	3.904020348	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=14/3584, ttl=5 (reply in 71)
58	3.904023767	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=15/3840, ttl=5 (reply in 72)
59	3.904032851	192.168.100.216	193.136.9.240	ICMP	74	Echo (ping) request id=0x3eal, seq=16/4096, ttl=6 (reply in 73)
60	3.904067558	192.168.100.216	192.168.100.216	TCP	182	Time-to-live exceeded (Time to live exceeded in transit)

Fig. 5: Identificação do endereço IP.

2.5 Exercício 2.b.

Questão

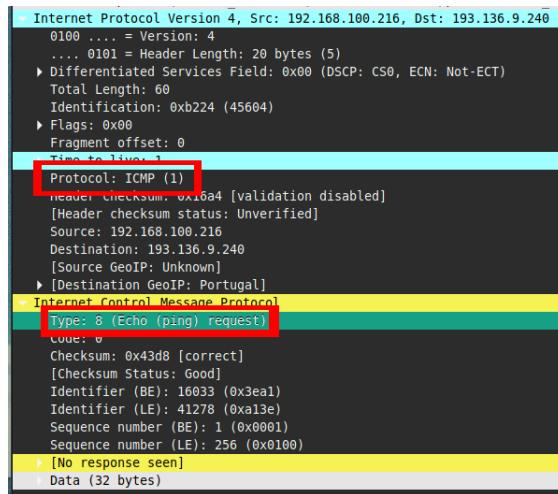
Qual é o valor do campo protocolo? O que identifica?

Resposta

O campo protocolo tem o valor "ICMP (1)". ICMP significa *Internet Control Message Protocol*. Este é utilizado para reportar erros no processamento de datagramas. Efetivamente, dentro dos possíveis erros temos, *destination unreachable* (quando o datagrama não consegue alcançar o destino), *time exceeded message* (quando um *gateway* processa um datagrama e descobre que o *TTL* é zero e tem que descartar o datagrama e consequentemente notificar o *host*), *echo request/reply* (quando são enviadas mensagens para funções de teste e controle da rede (*request*), caso a máquina esteja ligada responde com um *reply*) [1] [2]. Como as mensagens ICMP encontram-se ao nível de rede, estas são também elas encapsuladas em datagramas IP que, consequentemente, usam o protocolo IP.

Assim sendo, analisando a primeira mensagem ICMP, nomeadamente no separador do *Internet Protocol Version 4*, percebemos que se trata de uma mensagem que vem num protocolo *ICMP*. Além disso, é possível concluir que se trata de uma mensagem de *echo request*, se observarmos o campo *Type* no separador *Internet Control Message Protocol*. Desta forma, pode-se concluir que o computador usado para a resolução deste trabalho está a tentar perceber se consegue estabelecer uma ligação com o *host* marco.uminho.pt e para isso usa mensagens *ICMP* do tipo *echo request*.

Realização



```
Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.9.240
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 60
    Identification: 0xb224 (45604)
    Flags: 0x00
    Fragment offset: 0
    Time-to-live: 1
    Protocol: ICMP (1) Protocol: ICMP (1)
    Header checksum: 0xa13e [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.100.216
    Destination: 193.136.9.240
    [Source GeoIP: Unknown]
    [Destination GeoIP: Portugal]
  Internet Control Message Protocol
    Type: 8 (Echo (ping) request) Type: 8 (Echo (ping) request)
    code: 0
    Checksum: 0x43d8 [correct]
    [Checksum Status: Good]
    Identifier (BE): 16033 (0x3ea1)
    Identifier (LE): 41278 (0xa13e)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [No response seen]
    Data (32 bytes)
```

Fig. 6: Identificação do campo *Protocol*.

2.6 Exercício 2.c.

Questão

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

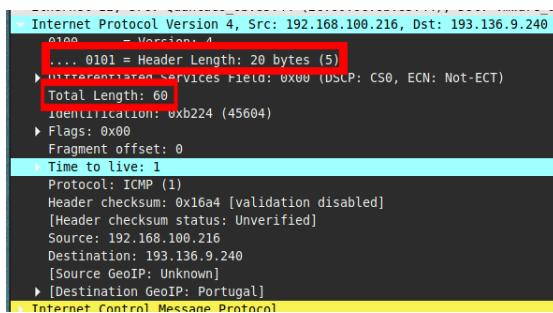
Resposta

O cabeçalho IPv4 tem 20 bytes.

O campo de dados (payload) do datagrama tem 40 bytes.

O cálculo do payload é feito retirando o tamanho do cabeçalho ao tamanho total do datagrama (60 bytes). Desta forma, basta fazer $60 - 20 = 40\text{bytes}$.

Realização



```
Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.9.240
  Version: 4
  Header Length: 20 bytes (5)
  Differentiation Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0xb224 (45604)
  Flags: 0x00
  Fragment offset: 0
  Time to live: 1
  Protocol: ICMP (1)
  Header checksum: 0x16a4 [validation disabled]
  [Header checksum status: Unverified]
  Source: 192.168.100.216
  Destination: 193.136.9.240
  [Source GeoIP: Unknown]
  [Destination GeoIP: Portugal]
  Internet Control Message Protocol
```

Fig. 7: Identificação do tamanho do *header* e do *payload*.

2.7 Exercício 2.d.

Questão

O datagrama IP foi fragmentado? Justifique.

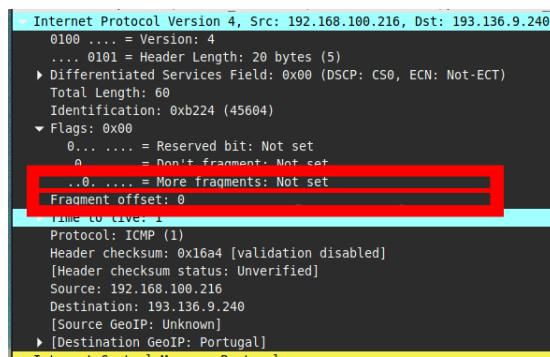
Resposta

A fragmentação acontece quando o tamanho total do datagrama excede o *MTU* disponível. Tendo em conta que por defeito o *traceroute* usa 60 bytes por datagrama e tem-se um *MTU* disponível de 1500 bytes, podemos conjeturar que não haverá fragmentação.

A verificação se um datagrama foi ou não fragmentado é feita com base em dois valores, o *fragment offset* (indica o *offset* em que o datagrama atual encaixa no datagrama original) e a *flag more fragments* (indica se existe mais fragmentos). Neste datagrama em específico o *fragment offset* = 0 e a *flag more fragments* = 0. Desta forma, tendo em conta o *fragment offset*, sabe-se que se o datagrama foi fragmentado então ele é necessariamente o primeiro. Além disso, se analisarmos a *flag more fragments* concluímos que para além do datagrama atual não existe mais nenhum associado a este.

Assim sendo, conjugando a informação dos dois parâmetros percebe-se que se o datagrama é o primeiro e não existe mais nenhum associado, então este é único e não foi fragmentado.

Realização



The screenshot shows an IP header from a packet captured by Wireshark. The 'Flags' field is expanded, showing the following bits:
0... = Reserved bit: Not set
0.... = Don't fragment: Not set
..0. = More fragments: Not set
Fragment offset: 0

Fig. 8: Fragmentação.

2.8 Exercício 2.e.

Questão

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 sua máquina. Para a sequência de mensagens ICMP enviadas pelo seu computador, indique que campos do cabeçalho IP variam de pacote para pacote.

Resposta

Os campos que vêm os seus valores alterados correspondem à *identification, header checksum e time to live (TTL)*.

A *identification* muda pois este campo identifica unicamente cada datagrama e visto que estes são sempre diferentes então o campo também o será.

O *header checksum* permite verificar que determinado header foi ou não corrompido. Desta forma o *checksum* identifica um determinado *header* num determinado estado. Assim sendo, o *checksum* muda pois este campo utiliza no seu algoritmo todas as palavras de 16 bits do *header* [1]. Efetivamente, sabendo que o *header*, propriamente dito, muda de datagrama para datagrama então o seu *checksum* também vai mudar.

O *TTL* muda pois a máquina que está a ser usada está a tentar contactar o *host* marco.uminho.pt. Começa por tentar com *TTL* = 1 e o pacote é descartado. Desta forma, vai aumentado o *TTL* até conseguir chegar ao *host* pretendido.

Realização

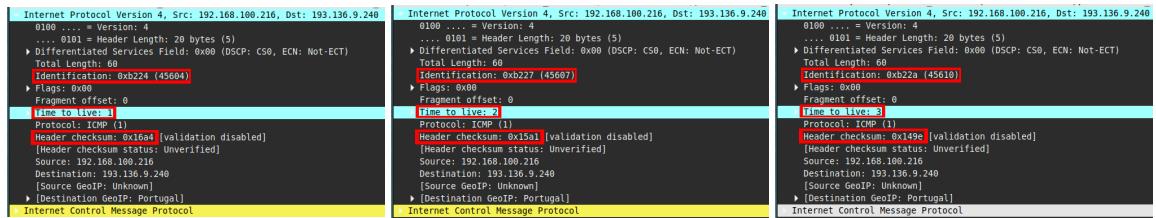


Fig. 9: Campos que mudam.

2.9 Exercício 2.f.

Questão

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

Resposta

O campo da indentificação corresponde a um valor que é incrementado e que identifica unicamente o datagrama em questão. Por exemplo, se o primeiro datagrama tiver *Identification* 0xb224 (45604), então o datagrama seguinte terá o valor 0xb225 (45605).

O TTL corresponde a uma variável que vai sendo decrementada sempre que é intersestada por um *router*. Visto que na primeira mensagem o TTL é 1, o datagrama é descartado imediatamente no primeiro *router*. Deste modo, é enviado, de seguida, um novo datagrama com TTL a 2 com a esperança de que este chegue desta vez ao destino. Caso não chegue, então no próximo datagrama o TTL será aumentado, e assim sucessivamente, até chegar ao destino.

Realização

A relação entre TTL pode ser observada na figura 12.

```
Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.9.240
 0100 .... = Version: 4
 .... 0101 = Header Length: 20 bytes (5)
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 60
 Identification: 0xb224 (45604) ■
 ▶ Flags: 0x00
 Fragment offset: 0
 ▶ Time to live: 1
 Protocol: ICMP (1)
 Header checksum: 0x16a4 [validation disabled]
 [Header checksum status: Unverified]
 Source: 192.168.100.216
 Destination: 193.136.9.240
 [Source GeoIP: Unknown]
 ▶ [Destination GeoIP: Portugal]
 ▶ Internet Control Message Protocol
```

(a) Primeiro datagrama.

```
Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.9.240
 0100 .... = Version: 4
 .... 0101 = Header Length: 20 bytes (5)
 ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
 Total Length: 60
 Identification: 0xb225 (45605) ■
 ▶ Flags: 0x00
 Fragment offset: 0
 ▶ Time to live: 1
 Protocol: ICMP (1)
 Header checksum: 0x16a3 [validation disabled]
 [Header checksum status: Unverified]
 Source: 192.168.100.216
 Destination: 193.136.9.240
 [Source GeoIP: Unknown]
 ▶ [Destination GeoIP: Portugal]
 ▶ Internet Control Message Protocol
```

(b) Datagrama seguinte.

Fig. 10: Identificação.

2.10 Exercício 2.g.

Questão

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?

Resposta

O IP 192.168.100.254 referencia a interface do primeiro router de acesso (visto ter os três primeiros campos iguais ao da máquina em que os testes estão a ser feitos e o último utiliza um número convencionado para ser utilizado para identificar a interface IP do router dentro da rede 192.168.100, na qual a máquina de testes se encontra).

Desta forma, quando analisamos os datagramas do 192.168.100.254 percebemos que estes têm todos o TTL = 64. O TTL toma um valor exageradamente elevado, devido ao desconhecimento que este tem da distância a que o *host* de destino se encontra. Por defeito, este router quando não tem informação da distância a que o *host* de destino se encontra envia datagramas com TTL = 64 e por isso todas os seus datagramas tem TTL igual.

Além disso, é possível concluir que as três mensagens recebidas deste router com *ICMP TTL exceeded* são a resposta ao envio feito pela máquina de testes de três datagramas de *echo (ping) request* com TTL apenas de 1. Estes datagramas chegaram ao *router* de acesso e ficaram com o TTL a 0 e a exceção veio enviado de no datagrama *ICMP TTL exceeded*.

Após estes datagramas é possível identificar um segundo conjunto de datagramas desta vez da interface IP 193.136.19.254. Pelo mesmo raciocínio podemos assumir que o IP desta interface identifica um *router*. Este novo *router* envia datagramas com TTL = 254 (constante). Este valor é considerável pela mesma razão explicitada anteriormente. Estes datagramas são a resposta a 3 datagramas enviados pela máquina de testes com TTL = 2, o que nos leva a concluir que é o segundo router por qual o nosso datagrama teve de passar para chegar ao marco.uminho.pt.

Realização

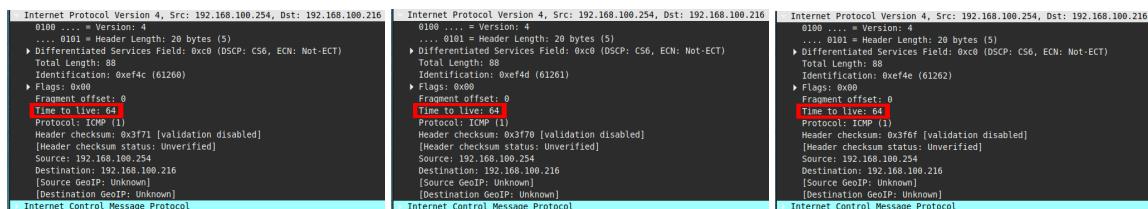


Fig. 11: Router de acesso com TTL = 64.

```

▼ Internet Protocol Version 4, Src: 193.136.19.254, Dst: 192.168.100.216
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x1147 (4423)
  ▶ Flags: 0x00
    Fragment offset: 0
  Time to live: 254
  Protocol: ICMP (1)
  Header checksum: 0xaafb6 [validation disabled]
  [Header checksum status: Unverified]
  Source: 193.136.19.254
  Destination: 192.168.100.216
  ▶ [Source GeoIP: Portugal]
  [Destination GeoIP: Unknown]
  ▶ Internet Control Message Protocol

▼ Internet Protocol Version 4, Src: 193.136.19.254, Dst: 192.168.100.216
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x1148 (4424)
  ▶ Flags: 0x00
    Fragment offset: 0
  Time to live: 254
  Protocol: ICMP (1)
  Header checksum: 0xaafb5 [validation disabled]
  [Header checksum status: Unverified]
  Source: 193.136.19.254
  Destination: 192.168.100.216
  ▶ [Source GeoIP: Portugal]
  [Destination GeoIP: Unknown]
  ▶ Internet Control Message Protocol

▼ Internet Protocol Version 4, Src: 193.136.19.254, Dst: 192.168.100.216
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  ▶ Differentiated Services Field: 0xc0 (DSCP: CS6, ECN: Not-ECT)
    Total Length: 56
    Identification: 0x1149 (4425)
  ▶ Flags: 0x00
    Fragment offset: 0
  Time to live: 254
  Protocol: ICMP (1)
  Header checksum: 0xaafb4 [validation disabled]
  [Header checksum status: Unverified]
  Source: 193.136.19.254
  Destination: 192.168.100.216
  ▶ [Source GeoIP: Portugal]
  [Destination GeoIP: Unknown]
  ▶ Internet Control Message Protocol

```

Fig. 12: Segundo router com TTL = 254.

2.11 Exercício 3.a.

Questão

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

Resposta

O pacote inicial tem um *total length* de 3033 bytes. O MTU (*Maximum Transmission Unit*) disponível é de apenas 1500 bytes e, como tal, o pacote inicial tem de ser fragmentado para que este possa ser transmitido na rede.

Realização

No.	Time	Source	Destination	Protocol	Length	Info
11	9.318600576	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xa8ca A d1-debug.dropbox.com
12	9.318606341	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xbc4 AAAA d1-debug.dropbox.com
13	9.319254879	192.168.100.254	192.168.100.216	DNS	190	Standard query response 0x3dcf AAAA d1-debug.dropbox.com CNAME block-debug.x.dro
14	9.649429727	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x3dcf A marco.uminho.pt
15	9.649437034	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x70d8 AAAA marco.uminho.pt
16	9.650136248	192.168.100.254	192.168.100.216	DNS	347	Standard query response 0x3dcf A marco.uminho.pt A 193.136.9.240 NS dns3.uminho.
17	9.656723657	192.168.100.254	192.168.100.216	DNS	129	Standard query response 0x70d8 AAAA marco.uminho.pt SOA dns.uminho.pt
18	9.656885085	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a813) [Reassembled in #20]
19	9.656889397	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a813) [Reassembled in #20]
*	20	9.656890359	192.168.100.216	193.136.9.240	ICMP	87 Echo (ping) request id=0x4af3, seq=1/256, ttl=1 (no response found!)
21	9.656896461	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a814) [Reassembled in #23]
22	9.656897423	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a814) [Reassembled in #23]
23	9.656898295	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=2/512, ttl=1 (no response found!)
24	9.656903585	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a815) [Reassembled in #26]
25	9.656904600	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a815) [Reassembled in #26]
26	9.656905558	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=3/768, ttl=1 (no response found!)
27	9.656909521	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a816) [Reassembled in #29]
28	9.656910428	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a816) [Reassembled in #29]
29	9.656911303	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=4/1024, ttl=2 (no response found!)
> Frame 18: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0						
> Ethernet II, Src: QuantaCo_6b:85:44 (2c:60:0c:6b:85:44), Dst: VMware_d2:19:f0 (00:0c:29:d2:19:f0)						
> Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.9.240						
0100 = Version: 4						
.... 0101 = Header Length: 20 bytes (5)						
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
Total Length: 1500						
Identification: 0xa813 (43027)						
> Flags: 0x01 (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: 0xfb14 [validation disabled]						
[Header checksum status: Unverified]						
Source: 192.168.100.216						
Destination: 193.136.9.240						
[Source GeoIP: Unknown]						

Fig. 13: A MTU é menor do que o tamanho do pacote.

2.12 Exercício 3.b.

Questão

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?

Resposta

A informação de que o datagrama foi fragmentado é dada através da *flag more fragments* que se encontra a 1 em conjugação com o campo *fragment offset*, que neste caso se encontra a 0. Desta forma, podemos concluir que este datagrama é efetivamente o primeiro (*fragment offset = 0*) de um PDU fragmentado (*flag more fragments = 1*). Este é um datagrama IP de 1480 bytes (MTU de 1500 bytes, dos quais 20 bytes servem de *header*).

Realização

No.	Time	Source	Destination	Protocol	Length	Info
11	9.318600576	192.168.100.216	192.168.100.254	DNS	88	Standard query 0xa8ca A dl-debug.dropbox.com
12	9.318606341	192.168.100.216	192.168.100.254	DNS	88	Standard query 0xbcd4 AAAA dl-debug.dropbox.com
13	9.319254879	192.168.100.254	192.168.100.216	DNS	198	Standard query response 0xbcd4 AAAA dl-debug.dropbox.com CNAME block-debug.x.drc
14	9.649429727	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x3dcf A marco.uminho.pt
15	9.649437034	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x78d8 AAAA marco.uminho.pt
16	9.650136248	192.168.100.254	192.168.100.216	DNS	347	Standard query response 0x3dcf A marco.uminho.pt A 193.136.9.240 NS dns3.uminho.pt
17	9.656723657	192.168.100.254	192.168.100.216	DNS	129	Standard query response 0x70d0 AAAA marco.uminho.pt SOA dns.uminho.pt
18	9.656885085	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a813) [Reassembled in #20]
19	9.656889397	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a813) [Reassembled in #20]
20	9.656890359	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request id=0x4af3, seq=1/256, ttl=1 (no response found!)
21	9.656896461	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a814) [Reassembled in #23]
22	9.656897423	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a814) [Reassembled in #23]
23	9.656898295	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request id=0x4af3, seq=2/512, ttl=1 (no response found!)
24	9.656903585	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a815) [Reassembled in #26]
25	9.656904600	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a815) [Reassembled in #26]
26	9.656905558	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request id=0x4af3, seq=3/768, ttl=1 (no response found!)
27	9.656909521	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a816) [Reassembled in #29]
28	9.656910428	192.168.100.216	193.136.9.248	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a816) [Reassembled in #29]
29	9.656911303	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request id=0x4af3, seq=4/1024, ttl=2 (no response found!)
0100 = Version: 4						
.... 0101 = Header Length: 20 bytes (5)						
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
Total Length: 1500						
Identification: 0xa813 (43027)						
Flags: 0x01 (More Fragments)						
.... = 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: 0xfb14 [validation disabled]						
[Header checksum status: Unverified]						
Source: 192.168.100.216						
Destination: 193.136.9.248						
[Source GeoIP: Unknown]						
[Destination GeoIP: Unknown]						
Reassembled IPv4 in frame: 20						
> Data (1480 bytes)						

Fig. 14: As flags e o fragment offset confirmam que é o primeiro fragmento.

2.13 Exercício 3.c.

Questão

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?

Resposta

Sabe-se que este não é o primeiro segmento pois o datagrama em questão tem *fragment offset* = 1480. Isto significa que, no datagrama original, este segmento começará a partir do byte 1480 do datagrama. Analisando a *flag more fragments*, percebe-se existem mais fragmentos, visto que esta é igual a 1.

Realização

No.	Time	Source	Destination	Protocol	Length	Info
11	9.318600576	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xa8ca A d1-debug.dropbox.com
12	9.318606341	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xbcd4 AAAA d1-debug.dropbox.com
13	9.319254879	192.168.100.254	192.168.100.216	DNS	190	Standard query response 0xbcda AAAA d1-debug.dropbox.com CNAME block-debug.x.dropbox.com
14	9.649429727	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x3dcf A marco.uminho.pt
15	9.649437034	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x70d8 AAAA marco.uminho.pt
16	9.650136248	192.168.100.254	192.168.100.216	DNS	347	Standard query response 0x3dcf A marco.uminho.pt A 193.136.9.240 NS dns3.uminho.pt
17	9.656723657	192.168.100.254	192.168.100.216	DNS	129	Standard query response 0x70d8 AAAA marco.uminho.pt SOA dns.uminho.pt
18	9.656885085	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=0, ID=a813) [Reassembled in #20]
19	9.656889397	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=1480, ID=a813) [Reassembled in #20]
20	9.656890359	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=1/256, ttl=1 (no response found!)
21	9.656896461	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=0, ID=a814) [Reassembled in #23]
22	9.656897423	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=1480, ID=a814) [Reassembled in #23]
23	9.656898295	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=2/512, ttl=1 (no response found!)
24	9.656903585	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=0, ID=a815) [Reassembled in #26]
25	9.656904600	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=1480, ID=a815) [Reassembled in #26]
26	9.656905558	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=3/768, ttl=1 (no response found!)
27	9.656909521	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=0, ID=a816) [Reassembled in #29]
28	9.656910428	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, offf=1480, ID=a816) [Reassembled in #29]
29	9.656911303	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=4/1024, ttl=2 (no response found!)
> Frame 19: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0						
> Ethernet II, Src: QuantaCo_6b:85:44 (2c:60:0c:6b:85:44), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)						
↳ Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.9.240						
0100 = Version: 4						
.... 0101 = Header Length: 20 bytes (5)						
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
Total Length: 1500						
Identification: 0xa813 (43027)						
> Flags: 0x01 (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: 0xfa5b [validation disabled]						
[Header checksum status: Unverified]						
Source: 192.168.100.216						
Destination: 193.136.9.240						
[Source GeoIP: Unknown]						

Fig. 15: As flags e o fragment offset confirmam que é o segundo fragmento.

2.14 Exercício 3.d.

Questão

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

Resposta

Analisando o parâmetro *identification* dos pacotes capturados pelo Wireshark, nota-se que existem três pacotes com o mesmo valor o que revela que pertencem ao mesmo PDU (*identification* = 0xa813). Desta forma podemos concluir que o datagrama foi fragmentado duas vezes resultando em três fragmentos. Os campos *fragment offset* e *more fragments* confirmam isto.

A deteção do último fragmento é feita através da flag *more fragments*. Se esta estiver a 1 então existem mais segmentos; por outras palavras, não é o último. Caso contrário, o segmento em questão é o último.

Realização

No.	Time	Source	Destination	Protocol	Length	Info
11	9.318600576	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xa8ca A dl-debug.dropbox.com
12	9.318606341	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xbcd4 AAAA dl-debug.dropbox.com
13	9.319254879	192.168.100.254	192.168.100.216	DNS	190	Standard query response 0xbcd4 AAAA dl-debug.dropbox.com CNAME block-debug.x.drc
14	9.649429727	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x3dcf A marco.uminho.pt
15	9.649437034	192.168.100.216	192.168.100.254	DNS	75	Standard query 0x70d8 AAAA marco.uminho.pt
16	9.650136248	192.168.100.254	192.168.100.216	DNS	347	Standard query response 0x3dcf A marco.uminho.pt A 193.136.9.240 NS dns3.uminho.
17	9.656723657	192.168.100.254	192.168.100.216	DNS	129	Standard query response 0x70d8 AAAA marco.uminho.pt SOA dns.uminho.pt
18	9.656885085	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a813) [Reassembled in #20]
19	9.656889397	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a813) [Reassembled in #20]
*	20	9.65689359	192.168.100.216	193.136.9.240	ICMP	87 Echo (ping) request id=0x4af3, seq=1/256, ttl=1 (no response found!)
21	9.656896461	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a814) [Reassembled in #23]
22	9.656897423	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a814) [Reassembled in #23]
23	9.656898295	192.168.100.216	193.136.9.240	ICMP	87 Echo (ping) request id=0x4af3, seq=2/512, ttl=1 (no response found!)	
24	9.656903585	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a815) [Reassembled in #26]
25	9.656904600	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a815) [Reassembled in #26]
26	9.656905558	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=3/768, ttl=1 (no response found!)
27	9.656909521	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, ID=a816) [Reassembled in #29]
28	9.656910428	192.168.100.216	193.136.9.240	IPv4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, ID=a816) [Reassembled in #29]
29	9.656911303	192.168.100.216	193.136.9.240	ICMP	87	Echo (ping) request id=0x4af3, seq=4/1024, ttl=2 (no response found!)
> Frame 19: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits) on interface 0						
> Ethernet II, Src: QuantaCo_6b:85:44 (2c:60:0c:6b:85:44), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)						
Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.9.240						
0100 = Version: 4						
.... 0101 = Header Length: 20 bytes (5)						
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
Total Length: 1500						
Identification: 0xa813 (43027)						
Flags: 0x01 (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: 0xfa5b [validation disabled]						
[Header checksum status: Unverified]						
Source: 192.168.100.216						
Destination: 193.136.9.240						
[Source GeoIP: Unknown]						

Fig. 16: A flag *more fragments* indica que é o último fragmento.

2.15 Exercício 3.e.

Questão

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.

Resposta

Os campos que mudam são *Total Length*, *more fragments* e *Fragment offset*.

No primeiro fragmento, o *Total Length* é igual ao MTU menos o *Header Length*, a flag *more fragments* é igual a 1 e o *Fragment offset* é igual a 0.

Entre o primeiro fragmento e o último fragmento não inclusives, o *Total Length* é igual ao MTU menos o *Header Length*, a flag *more fragments* é igual a 1 e o *Fragment offset* é igual a $(\text{MTU} - \text{Header Length}) * (n - 1)$, n sendo o n-ésimo fragmento do datagrama (n = 1, 2, ...).

No último fragmento, o *Total Length* é menor ou igual ao MTU, a flag *more fragments* é igual a 0 e o *Fragment offset* é igual a $(\text{MTU} - \text{Header Length}) * (n - 1)$, n sendo o número de fragmentos do datagrama.

Realização

Verifica-se no datagrama analisado que as *flags* e o *fragment offset* evoluem como indicado [14][15][16]

3 Parte II - Endereçamento e Encaminhamento IP

3.1 Exercício 2.1.a

Questão

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.

Resposta

A máscara de rede utilizada é 255.255.255.0. Isto deve-se a facto de todos os IPv4 atribuídos aos diferentes componentes da rede terem /24 como quantidade de bits que identificam a rede.

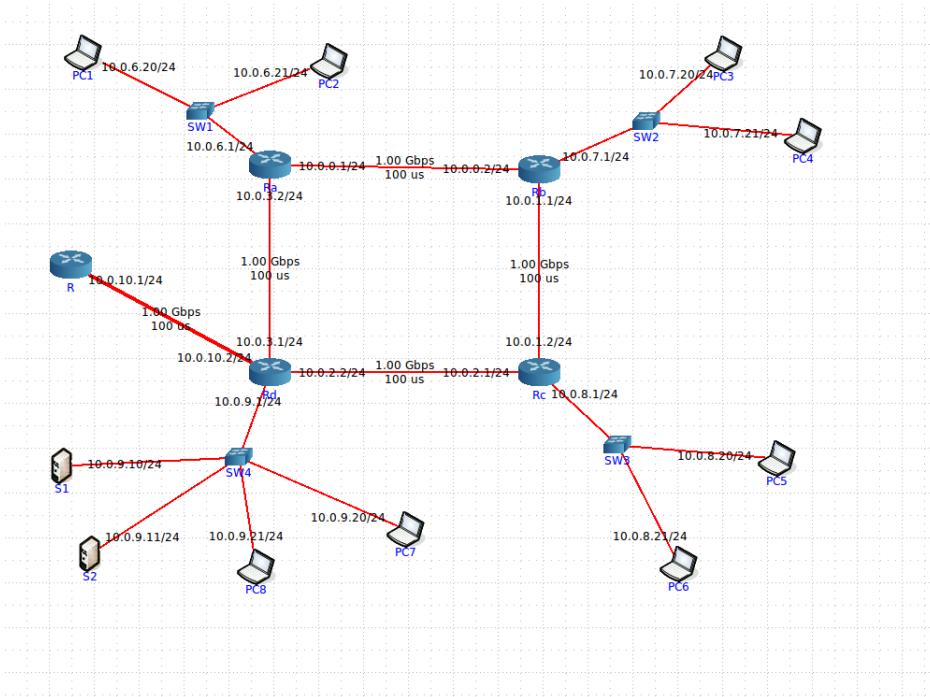


Fig. 17: Topologia da rede.

3.2 Exercício 2.1.b

Questão

Tratam-se de endereços públicos ou privados? Porquê?

Resposta

Tratam-se de endereços privados pois o primeiro octeto é 00001010, que em decimal representa 10. Sendo este o primeiro octeto podemos afirmar que é um endereço IP privado visto que pertence ao conjunto de prefixos que identificam um endereçamento privado. Os conjuntos de endereços privados são os seguintes:

- Start address: 10.0.0.0 → End Address: 10.255.255.255
- Start address: 172.16.0.0 → End Address: 172.31.255.255
- Start address: 192.168.0.0 → End Address: 192.168.255.255
- Start address: 169.254.0.0 → End Address: 169.254.255.255

Realização

Pela observação da topologia da rede 17.

3.3 Exercício 2.1.c

Questão

Porque razão não é atribuído um endereço IP aos switches?

Resposta

Na topologia em questão se vier um pacote com destino ao PC7, este vem com endereço de destino de 10.0.9.20/24. Para tal, o pacote vem do *router* R e chega ao *router* Rd. O *router* Rd aplica a NetMask 255.255.255.0 ao endereço de destino e obtém 10.0.9.0. Consegue fazer match no endereço 10.0.9.1 e envia o pacote para o SW4. Ao chegar ao SW4, este avalia o endereço MAC de destino e duas situações podem acontecer. Se este tiver nos seus registos o endereço, então envia diretamente ao host especificado. Caso não tenha ainda essa informação envia para todos os hosts a ele ligados.

Efetivamente, os *switches* não operam ao nível de rede e como tal a camada de rede, nomeadamente o IP, é completamente transparente para estes dispositivos.

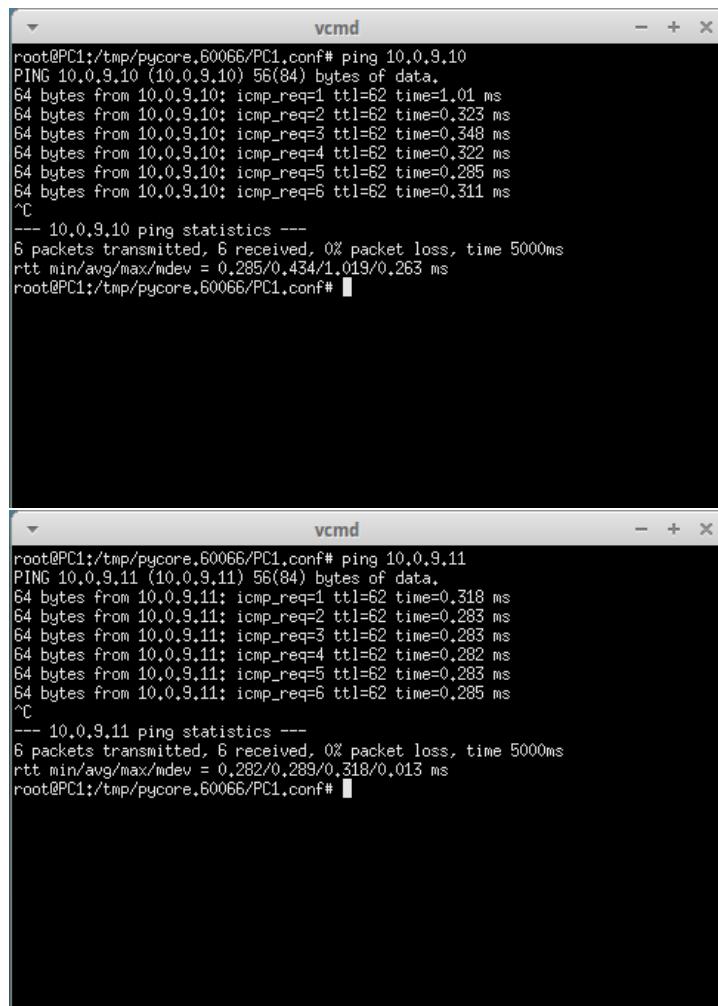
Através do exemplo a cima podemos concluir que não é necessário que este tenha IP pois os pacotes são enviados para o *switch* com base no IP da interface de saída do *router*. Após o datagrama chegar ao *switch* este quanto muito usa endereços MAC por forma a fazer *forward* do datagrama para o *host* respetivo.

3.4 Exercício 2.1.d

Questão

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

Resposta



The image shows two terminal windows titled "vcmd" running on a Linux system. Both windows display the output of a "ping" command from a root shell.

The top window shows the output of:

```
root@PC1:/tmp/pycore.60066/PC1.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_req=1 ttl=62 time=1.01 ms
64 bytes from 10.0.9.10: icmp_req=2 ttl=62 time=0.323 ms
64 bytes from 10.0.9.10: icmp_req=3 ttl=62 time=0.348 ms
64 bytes from 10.0.9.10: icmp_req=4 ttl=62 time=0.322 ms
64 bytes from 10.0.9.10: icmp_req=5 ttl=62 time=0.285 ms
64 bytes from 10.0.9.10: icmp_req=6 ttl=62 time=0.311 ms
^C
--- 10.0.9.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.285/0.434/1.019/0.263 ms
root@PC1:/tmp/pycore.60066/PC1.conf#
```

The bottom window shows the output of:

```
root@PC1:/tmp/pycore.60066/PC1.conf# ping 10.0.9.11
PING 10.0.9.11 (10.0.9.11) 56(84) bytes of data.
64 bytes from 10.0.9.11: icmp_req=1 ttl=62 time=0.318 ms
64 bytes from 10.0.9.11: icmp_req=2 ttl=62 time=0.283 ms
64 bytes from 10.0.9.11: icmp_req=3 ttl=62 time=0.283 ms
64 bytes from 10.0.9.11: icmp_req=4 ttl=62 time=0.282 ms
64 bytes from 10.0.9.11: icmp_req=5 ttl=62 time=0.283 ms
64 bytes from 10.0.9.11: icmp_req=6 ttl=62 time=0.285 ms
^C
--- 10.0.9.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.282/0.289/0.318/0.013 ms
root@PC1:/tmp/pycore.60066/PC1.conf#
```

Fig. 18: Teste de conectividade entre PC1 e os servidores S1 (à esquerda) e S2 (à direita).

The image displays two terminal windows titled "vcmd" running on a Linux system. Both windows show the output of a "ping" command to an IP address.

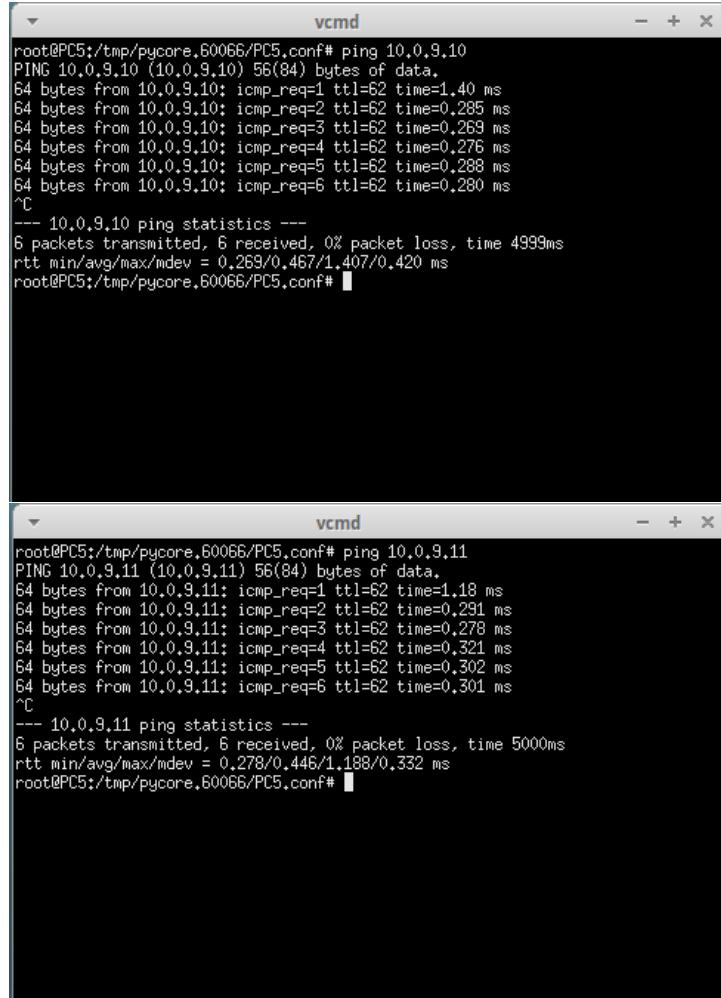
The top window shows the output of:

```
root@PC3:/tmp/pycore.60066/PC3.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_req=1 ttl=61 time=1.62 ms
64 bytes from 10.0.9.10: icmp_req=2 ttl=61 time=0.540 ms
64 bytes from 10.0.9.10: icmp_req=3 ttl=61 time=0.509 ms
64 bytes from 10.0.9.10: icmp_req=4 ttl=61 time=0.525 ms
64 bytes from 10.0.9.10: icmp_req=5 ttl=61 time=0.547 ms
64 bytes from 10.0.9.10: icmp_req=6 ttl=61 time=0.534 ms
^C
--- 10.0.9.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5001ms
rtt min/avg/max/mdev = 0.509/0.713/1.627/0.409 ms
root@PC3:/tmp/pycore.60066/PC3.conf#
```

The bottom window shows the output of:

```
root@PC3:/tmp/pycore.60066/PC3.conf# ping 10.0.9.11
PING 10.0.9.11 (10.0.9.11) 56(84) bytes of data.
64 bytes from 10.0.9.11: icmp_req=1 ttl=61 time=0.835 ms
64 bytes from 10.0.9.11: icmp_req=2 ttl=61 time=0.524 ms
64 bytes from 10.0.9.11: icmp_req=3 ttl=61 time=0.576 ms
64 bytes from 10.0.9.11: icmp_req=4 ttl=61 time=0.523 ms
64 bytes from 10.0.9.11: icmp_req=5 ttl=61 time=0.524 ms
64 bytes from 10.0.9.11: icmp_req=6 ttl=61 time=0.546 ms
^C
--- 10.0.9.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.523/0.588/0.835/0.112 ms
root@PC3:/tmp/pycore.60066/PC3.conf#
```

Fig. 19: Teste de conectividade entre PC3 e os servidores S1 (à esquerda) e S2 (à direita).



The image shows two terminal windows titled "vcmd" running on a Linux system. Both windows display the output of a "ping" command from a host with IP 10.0.9.10 to two different destinations: 10.0.9.10 and 10.0.9.11. The output shows six ICMP requests being sent to each destination, with their respective times and packet sizes. The results indicate 0% packet loss and provide statistics for the round-trip time (RTT) and its variance.

```
root@PC5:/tmp/pycore.60066/PC5.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_req=1 ttl=62 time=1.40 ms
64 bytes from 10.0.9.10: icmp_req=2 ttl=62 time=0.285 ms
64 bytes from 10.0.9.10: icmp_req=3 ttl=62 time=0.269 ms
64 bytes from 10.0.9.10: icmp_req=4 ttl=62 time=0.276 ms
64 bytes from 10.0.9.10: icmp_req=5 ttl=62 time=0.288 ms
64 bytes from 10.0.9.10: icmp_req=6 ttl=62 time=0.280 ms
^C
--- 10.0.9.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.269/0.467/1.407/0.420 ms
root@PC5:/tmp/pycore.60066/PC5.conf# 

root@PC5:/tmp/pycore.60066/PC5.conf# ping 10.0.9.11
PING 10.0.9.11 (10.0.9.11) 56(84) bytes of data.
64 bytes from 10.0.9.11: icmp_req=1 ttl=62 time=1.18 ms
64 bytes from 10.0.9.11: icmp_req=2 ttl=62 time=0.291 ms
64 bytes from 10.0.9.11: icmp_req=3 ttl=62 time=0.278 ms
64 bytes from 10.0.9.11: icmp_req=4 ttl=62 time=0.321 ms
64 bytes from 10.0.9.11: icmp_req=5 ttl=62 time=0.302 ms
64 bytes from 10.0.9.11: icmp_req=6 ttl=62 time=0.301 ms
^C
--- 10.0.9.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.278/0.446/1.198/0.332 ms
root@PC5:/tmp/pycore.60066/PC5.conf# 
```

Fig. 20: Teste de conectividade entre PC5 e os servidores S1 (à esquerda) e S2 (à direita).

```

vcmd
root@PC7:/tmp/pycore.60066/PC7.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_req=1 ttl=64 time=0.063 ms
64 bytes from 10.0.9.10: icmp_req=2 ttl=64 time=0.028 ms
64 bytes from 10.0.9.10: icmp_req=3 ttl=64 time=0.028 ms
64 bytes from 10.0.9.10: icmp_req=4 ttl=64 time=0.028 ms
64 bytes from 10.0.9.10: icmp_req=5 ttl=64 time=0.035 ms
64 bytes from 10.0.9.10: icmp_req=6 ttl=64 time=0.028 ms
^C
--- 10.0.9.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4999ms
rtt min/avg/max/mdev = 0.028/0.035/0.063/0.012 ms
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf#
root@PC7:/tmp/pycore.60066/PC7.conf# ■

vcmd
root@PC7:/tmp/pycore.60066/PC7.conf# ping 10.0.9.11
PING 10.0.9.11 (10.0.9.11) 56(84) bytes of data.
64 bytes from 10.0.9.11: icmp_req=1 ttl=64 time=0.053 ms
64 bytes from 10.0.9.11: icmp_req=2 ttl=64 time=0.024 ms
64 bytes from 10.0.9.11: icmp_req=3 ttl=64 time=0.023 ms
64 bytes from 10.0.9.11: icmp_req=4 ttl=64 time=0.032 ms
64 bytes from 10.0.9.11: icmp_req=5 ttl=64 time=0.027 ms
64 bytes from 10.0.9.11: icmp_req=6 ttl=64 time=0.028 ms
64 bytes from 10.0.9.11: icmp_req=7 ttl=64 time=0.028 ms
^C
--- 10.0.9.11 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 5996ms
rtt min/avg/max/mdev = 0.023/0.030/0.053/0.011 ms
root@PC7:/tmp/pycore.60066/PC7.conf# ■

```

Fig. 21: Teste de conectividade entre PC7 e os servidores S1 (à esquerda) e S2 (à direita).

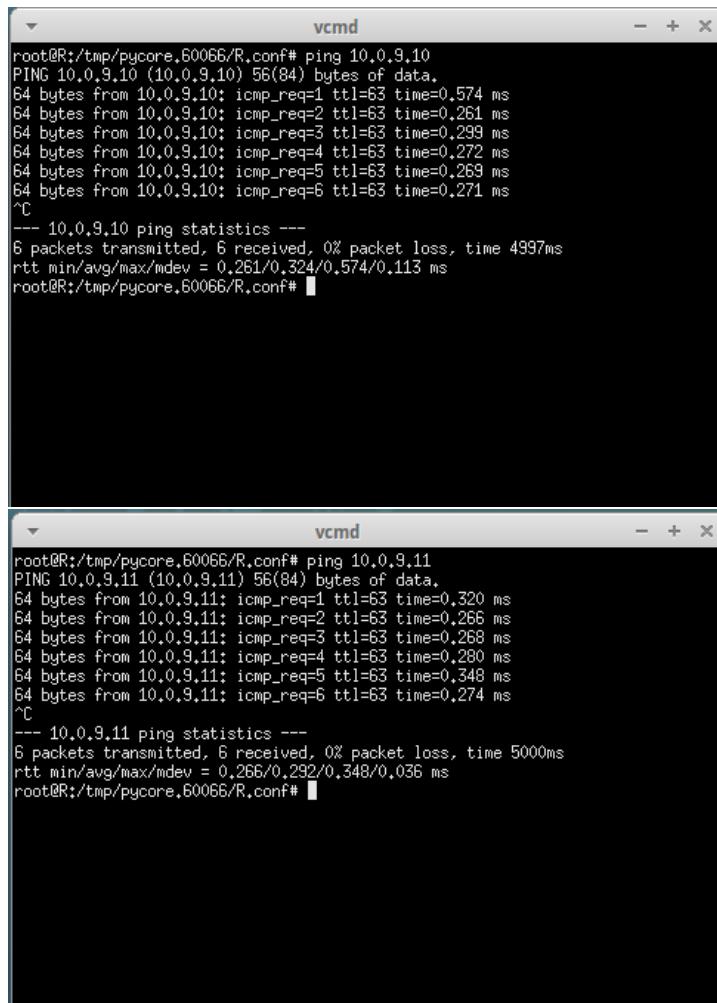
Tendo em conta os resultados do comando *ping* pode-se afirmar que existe conectividade entre os *hosts* dos diferentes departamentos e os servidores do departamento D.

3.5 Exercício 2.1.e

Questão

Verifique se existe conectividade IP do router de acesso para os servidores S1 e S2.

Resposta



The figure displays two terminal windows titled "vcmd" running on a router (R). Both windows show the output of a "ping" command to a server at 10.0.x.y, where x is either 9.10 or 9.11, and y is either 10 or 11. The top window shows the ping results for 10.0.9.10, and the bottom window shows the results for 10.0.9.11. Both outputs show six ICMP requests being sent with TTL=63, and the responses are received back at the source. The statistics at the end of each ping session indicate 0% packet loss and round-trip times (rtt) ranging from approximately 0.261 ms to 0.348 ms.

```
root@R:/tmp/pycore.60066/R.conf# ping 10.0.9.10
PING 10.0.9.10 (10.0.9.10) 56(84) bytes of data.
64 bytes from 10.0.9.10: icmp_req=1 ttl=63 time=0.574 ms
64 bytes from 10.0.9.10: icmp_req=2 ttl=63 time=0.261 ms
64 bytes from 10.0.9.10: icmp_req=3 ttl=63 time=0.299 ms
64 bytes from 10.0.9.10: icmp_req=4 ttl=63 time=0.272 ms
64 bytes from 10.0.9.10: icmp_req=5 ttl=63 time=0.269 ms
64 bytes from 10.0.9.10: icmp_req=6 ttl=63 time=0.271 ms
^C
--- 10.0.9.10 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 4997ms
rtt min/avg/max/mdev = 0.261/0.324/0.574/0.113 ms
root@R:/tmp/pycore.60066/R.conf#
```



```
root@R:/tmp/pycore.60066/R.conf# ping 10.0.9.11
PING 10.0.9.11 (10.0.9.11) 56(84) bytes of data.
64 bytes from 10.0.9.11: icmp_req=1 ttl=63 time=0.320 ms
64 bytes from 10.0.9.11: icmp_req=2 ttl=63 time=0.266 ms
64 bytes from 10.0.9.11: icmp_req=3 ttl=63 time=0.268 ms
64 bytes from 10.0.9.11: icmp_req=4 ttl=63 time=0.280 ms
64 bytes from 10.0.9.11: icmp_req=5 ttl=63 time=0.348 ms
64 bytes from 10.0.9.11: icmp_req=6 ttl=63 time=0.274 ms
^C
--- 10.0.9.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5000ms
rtt min/avg/max/mdev = 0.266/0.292/0.348/0.036 ms
root@R:/tmp/pycore.60066/R.conf#
```

Fig. 22: Teste de conectividade entre R e os servidores S1 (à esquerda) e S2 (à direita).

Tendo em conta os resultados obtidos em 33, podemos afirmar que existe conectividade entre o *router* R e os servidores S1 e S2.

3.6 Exercício 2.2.a

Questão

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).

Resposta

A primeira linha na tabela de endereçamento do PC3 representa a rota por defeito, isto é, quando entre no PC3 um datagrama com um IP que não faz *match* com nenhum IP da coluna *Destination* então este é reencaminhado para o *router* de acesso, ou seja, 10.0.7.1 ficando este responsável pelo seu encaminhamento. Denotando que a primeira linha, por ser um endereço estático, tem a menor prioridade. Esta linha faz *match* com qualquer endereço pois tem uma *Genmask* igual a 0.0.0.0. Efetivamente, quando chega à máquina PC3, por exemplo, um datagrama com o IP 10.0.8.20, este é comparado com todas as linhas da tabela. Posteriormente é aplicada a máscara obtendo-se 0.0.0.0. O resultado é depois comparado com o campo *Destination* e faz *match*. De seguida é enviado para o *router* de acesso explicitado pelo campo *Gateway*.

A segunda linha é responsável por encaminhar, através da própria máquina, o trânsito local, ou seja, o próprio PC3 encaminha os datagramas para *hosts* vizinhos. Deste modo, todos os datagramas que tiverem como prefixo de rede 10.0.7 vão fazer *match* na segunda linha, isto porque, chegando por exemplo o datagrama com o IP 10.0.7.21, é aplicada a máscara 255.255.255.0 e resulta a *destination* 10.0.7.0. Posteriormente é enviado para o endereço dito no *Gateway*, 0.0.0.0, ou seja, fica responsável por encaminhar este datagrama a própria máquina. Este processo de encaminhamento resulta pois os *hosts* tem conhecimento da topologia da rede local.

A tabela de encaminhamento do *router* Rb não inclui rotas por defeito pois todas as rotas existentes na rede encontram-se identificadas na tabela à partida.

Através da análise da topologia na rede criada com o CORE percebemos que o *router* Rb deverá ser responsável por encaminhar para a rede 10.0.0.0, 10.0.1.0, e 10.0.7.0. A tabela de encaminhamento confirma as nossas suspeitas, pois todas as linhas que têm os endereços IP atrás indicados têm como *Gateway* 0.0.0.0, que significa que o próprio *router* fica responsável por encaminhar esses datagramas.

Todas as outras redes existentes encontram-se na tabela com o endereço de encaminhamento apropriado. Nomeadamente, caso os datagramas tenham como destino a rede 10.0.x.0, em que x=2,3,6,8,9,10, estes são reencaminhados para uma das duas redes a que o *router* Rb tem acesso, isto é, 10.0.0.1 ou 10.0.1.2. Estas entradas em específico na tabela têm a flag G no campo *Flags*, visto não ser um encaminhamento direto.

Realização

The image displays two terminal windows side-by-side, both titled "vcmd".

The left terminal window shows the routing table for PC3:

```
root@PC3:/tmp/pycore.60066/PC3.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
0.0.0.0         10.0.7.1       0.0.0.0       UG      0 0          0 eth0
10.0.7.0        0.0.0.0       255.255.255.0 U        0 0          0 eth0
root@PC3:/tmp/pycore.60066/PC3.conf#
```

The right terminal window shows the routing table for Rb:

```
root@Rb:/tmp/pycore.60066/Rb.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0         0.0.0.0       255.255.255.0 U        0 0          0 eth0
10.0.1.0         0.0.0.0       255.255.255.0 U        0 0          0 eth1
10.0.2.0         10.0.1.2       255.255.255.0 UG      0 0          0 eth1
10.0.3.0         10.0.0.1       255.255.255.0 UG      0 0          0 eth0
10.0.6.0         10.0.0.1       255.255.255.0 UG      0 0          0 eth0
10.0.7.0         0.0.0.0       255.255.255.0 U        0 0          0 eth2
10.0.8.0         10.0.1.2       255.255.255.0 UG      0 0          0 eth1
10.0.9.0         10.0.0.1       255.255.255.0 UG      0 0          0 eth0
10.0.10.0        10.0.0.1      255.255.255.0 UG      0 0          0 eth0
root@Rb:/tmp/pycore.60066/Rb.conf#
```

Fig. 23: Tabelas de encaminhamento do PC3 (à esquerda) e do Rb (à direita).

3.7 Exercício 2.2.b

Questão

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

Resposta

A rede implementada no CORE tem, efetivamente, os dois tipos de encaminhamento, isto é, estático e dinâmico.

Em cada uma das tabelas de encaminhamento dos diferentes *hosts* que constituem a rede apenas são visíveis duas rotas que se mantém ao longo do tempo. Essas entradas da tabela representam, nomeadamente, a rota por defeito e a rota de tráfego da rede interna, que são sempre rotas estáticas.

Na rede core da topologia, isto é, no anel de rede criado entre os *routers* Ra, Rb, Rc e Rb é implementado encaminhamento dinâmico através do protocolo OSPF. Este protocolo é usado para realizar o reconhecimento da rede e consequentemente definir as rotas que os pacotes devem tomar.

Desta forma, é possível reparar que mal iniciamos o fluxo na rede no emulador *CORE*, a tabela de encaminhamento dos *routers* encontram-se apenas com três rotas predefinidas (estáticas), correspondentes às três interfaces que cada *router* interliga. Passados alguns minutos é possível observar que a tabela de encaminhamento obtém novas entradas (referência da imagem com as duas tabelas do netstat). Esta observação leva-nos a pensar que as novas entradas possam ser resultado de encaminhamento dinâmico. De forma a provar a veracidade dessa conjectura, tentou-se perceber se dada alguma falha na rede, por exemplo, com um *router* a desligar uma das suas interfaces, se esta consegue ajustar as diferentes rotas de encaminhamento de forma a suprimir o erro ocorrido.

Assim sendo, utilizou-se o PC3 e o S1, como componentes sobre os quais se averiguaria o comportamento dos pacotes quando feito um *ping* do *host* PC3 para o *host* S1. O comando *ping* é realizado às **10:19:00h**. Consequentemente é possível perceber que, ao analisar o tráfego capturado pelo *Wireshark* no IP 10.0.3.2 da interface do *router* Ra, que a partir dessa mesma hora (**10:19:00h**) existe passagem de pacotes de "ida" no *router* Ra, isto é, pacotes que têm como *source* o IP 10.0.7.20 e como *destination* o IP 10.0.9.10 [24]. Da mesma forma, se analisarmos o tráfego capturado pelo *Wireshark* no IP 10.0.1.2 do *router* Rc, percebe-se que por volta da mesma hora (**10:19:00h**) existe pacotes ICMP com sentido contrário, isto é, pacotes que têm como *source* o IP 10.0.9.10 e como *destination* o IP 10.0.7.20 [24]. Após esta análise conclui-se que o tráfego se encontra normalizado e é realizado pelo trajeto apresentado na imagem [25]. Efetivamente, a verde está demarcado o percurso intitulado de "ida", ou seja, percurso pelo qual passam os pacotes enviados do *host* PC3 para o *host* S1. A laranja temos o percurso de "volta", ou seja, é o percurso pelo qual passam os pacotes enviados de S1 para o PC3.

16	22.486136	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:01,454684
17	23.487572	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:02,456120
18	24.489979	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:03,458527
19	25.489384	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:04,457932
20	26.489358	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:05,457906
21	27.489297	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:06,457845
22	28.489374	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:07,457922

(a) Pacotes ICMP no percurso de "ida".

9	10.510695	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:01,455745
10	11.511321	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:02,456371
11	12.513739	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:03,458789
12	13.513147	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:04,458197
13	14.513108	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:05,458158
14	15.513046	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:06,458096

(b) Pacotes ICMP no percurso de "volta".

Fig. 24: Pacotes ICMP a circular na rede.

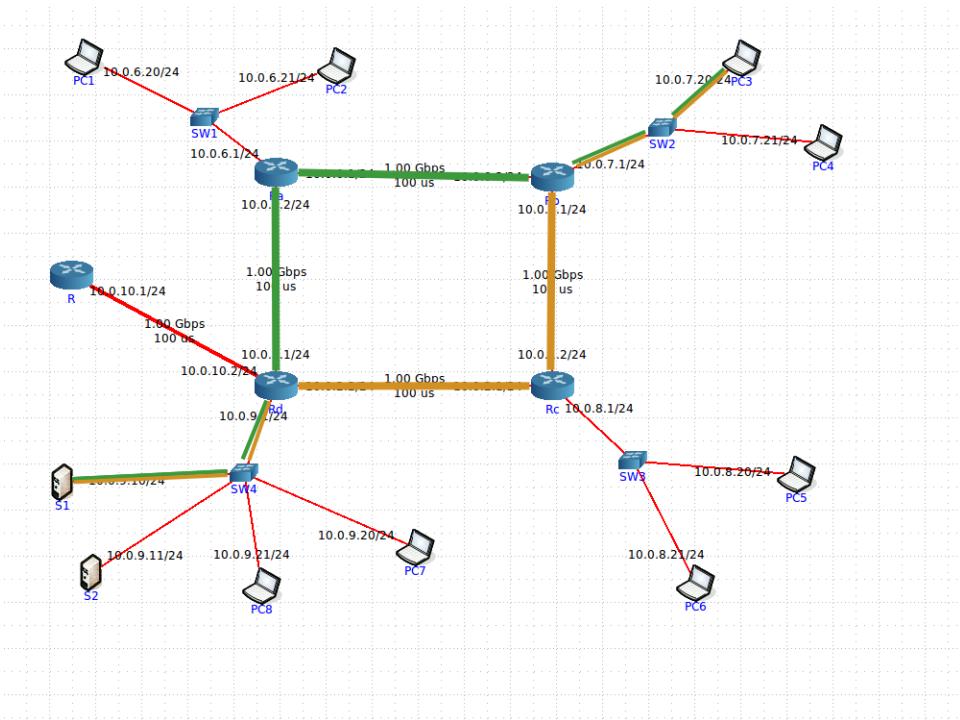


Fig. 25: Percurso de "ida" (a verde) e percurso de "volta" (a laranja).

De seguida, tentou-se realizar uma perturbação no percurso anteriormente estabelecido de forma a provocar ajustes nas rotas de encaminhamento definidas até ao momento pelos diversos *routers*. Para tal, colocou-se a interface eth0, que corresponde ao IP 10.0.0.1, do *router Ra down*, com o comando **ifconfig eth0 down**. Desta forma, os pacotes que vão do PC3 para o S1 deixam de conseguir seguir a rota definida até ao momento. Esta ação foi realizada às **10:19:14h** [26].

```
root@Ra:/tmp/pycore.59148/Ra.conf# date +%T && ifconfig eth0 down
10:19:14
```

Fig. 26: Desligada a interface eth0 do *router Ra*.

Consequentemente, por volta das **10:19:14h**, no tráfego capturado no *Wireshark* das interfaces dos dois *routers*, surge uma afluência significativa de pacotes com o protocolo OSPF [27]. Efetivamente, isto acontece pois como houve uma perturbação na rede, os *routers* precisam de entender que parte da rede ficou afetada com essa perturbação e como podem solucionar o problema (caso ele existe, que é o caso). Para tal usam o **protocolo OSPF** para recalcular e atualizar a tabela de encaminhamento de forma a ajustarem-se à mudança que ocorreu. Como resultado à alteração por nós imposta, o *router Rb* atualiza a sua tabela de encaminhamento de acordo com a imagem 28.

35	35.278392	10.0.3.2	224.0.0.5	OSPF	206 LS Update	10:19:14,246940
36	35.378227	fe80::200:ff:fea:7	ff02::5	OSPF	114 LS Update	10:19:14,346775
37	36.032609	10.0.3.1	224.0.0.5	OSPF	78 LS Acknowledge	10:19:15,001157
38	38.381576	fe80::200:ff:fea:6	ff02::5	OSPF	90 LS Acknowledge	10:19:17,350124
39	41.925941	fe80::200:ff:fea:7	ff02::5	OSPF	94 Hello Packet	10:19:20,894489
40	41.934811	10.0.3.2	224.0.0.5	OSPF	82 Hello Packet	10:19:20,903359
41	41.990343	10.0.3.2	10.0.3.1	OSPF	110 LS Update	10:19:20,958891
42	41.991640	10.0.3.1	224.0.0.5	OSPF	82 Hello Packet	10:19:20,960188
43	42.036716	10.0.3.1	224.0.0.5	OSPF	78 LS Acknowledge	10:19:21,005264

(a) Troca de pacotes OSPF na rota de "ida" (linhas a preto).

28	23.302349	10.0.1.2	224.0.0.5	OSPF	110 LS Update	10:19:14,247399
29	23.603330	fe80::200:ff:fea:3	ff02::5	OSPF	114 LS Update	10:19:14,548380
30	24.046206	10.0.1.1	224.0.0.5	OSPF	78 LS Acknowledge	10:19:14,991256
31	24.512847	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:15,457897	
32	24.513118	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:15,458168	
33	25.512852	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:16,457902	
34	25.513115	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:16,458165	
35	26.512865	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:17,457915	
36	26.513124	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:17,458174	
37	26.606872	fe80::200:ff:fea:2	ff02::5	OSPF	90 LS Acknowledge	10:19:17,551922
38	27.512811	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:18,457861	
39	27.513080	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:18,458130	
40	28.513665	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:19,458715	
41	28.513932	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:19,458982	
42	29.512847	10.0.7.20	10.0.9.10	ICMP	98 Echo (ping) requ 10:19:20,457897	
43	29.513099	10.0.9.10	10.0.7.20	ICMP	98 Echo (ping) repl 10:19:20,458149	
44	29.516820	00:00:00_aa:00:02	00:00:00_aa:00:03	ARP	42 Who has 10.0.1.2 10:19:20,461870	
45	29.516837	00:00:00_aa:00:03	00:00:00_aa:00:02	ARP	42 10.0.1.2 is at 0 10:19:20,461887	
46	29.980417	fe80::200:ff:fea:2	ff02::5	OSPF	94 Hello Packet	10:19:20,925467
47	30.000440	10.0.1.1	224.0.0.5	OSPF	82 Hello Packet	10:19:20,945490
48	30.014219	10.0.1.2	224.0.0.5	OSPF	110 LS Update	10:19:20,959269
49	30.025399	10.0.1.2	224.0.0.5	OSPF	82 Hello Packet	10:19:20,970449
50	30.050896	10.0.1.1	224.0.0.5	OSPF	78 LS Acknowledge	10:19:20,995946

(b) Troca de pacotes OSPF na rota de "volta" (linhas a preto).

Fig. 27: Pacotes ICMP a circular na rede.

```

vcmd
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        0.0.0.0        255.255.255.0 U        0 0          0 eth0
10.0.1.0        0.0.0.0        255.255.255.0 U        0 0          0 eth1
10.0.2.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.3.0        10.0.0.1       255.255.255.0 UG       0 0          0 eth0
10.0.6.0        10.0.0.1       255.255.255.0 UG       0 0          0 eth0
10.0.7.0        0.0.0.0        255.255.255.0 U        0 0          0 eth2
10.0.8.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.9.0        10.0.0.1       255.255.255.0 UG       0 0          0 eth0
10.0.10.0       10.0.0.1       255.255.255.0 UG       0 0          0 eth0
root@Rb:/tmp/pycore.40591/Rb.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        0.0.0.0        255.255.255.0 U        0 0          0 eth0
10.0.1.0        0.0.0.0        255.255.255.0 U        0 0          0 eth1
10.0.2.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.3.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.6.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.7.0        0.0.0.0        255.255.255.0 U        0 0          0 eth2
10.0.8.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.9.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.10.0       10.0.1.2       255.255.255.0 UG       0 0          0 eth1
root@Rb:/tmp/pycore.40591/Rb.conf#

```

Fig. 28: Atualização da tabela de encaminhamento de Rb.

De seguida, as tabelas de encaminhamento encontram-se atualizadas e o tráfego normaliza. Através da análise do tráfego capturado pelo *Wireshark* percebe-se que este apenas é feito pelo *router Rc*, ou seja, tanto os pacotes enviados do PC3 para o S1, como os pacotes enviados do S1 para o PC3 utilizam a mesma rota [29].

35	35.278393	10.0.3.2	224.0.0.5	OSPF	266	LS Update	10:19:14.246940
36	35.143327	fe80::200:ff:fea:7	ff02::1:9	OSPF	117	LS Update	10:19:14.246975
37	35.037689	10.0.3.1	224.0.0.5	OSPF	78	LS Acknowledge	10:19:15.001157
38	35.301576	fe80::200:ff:fea:6	ff02::5	OSPF	90	LS Acknowledge	10:19:17.350124
39	41.925041	fe80::200:ff:fea:7	ff02::5	OSPF	94	Hello Packet	10:19:20.894489
40	41.934011	10.0.3.2	224.0.0.5	OSPF	82	Hello Packet	10:19:20.903359
41	41.998343	10.0.3.2	10.0.3.1	OSPF	110	LS Update	10:19:20.958891
42	41.931648	10.0.3.1	224.0.0.5	OSPF	82	Hello Packet	10:19:20.960188
43	42.036716	10.0.3.1	224.0.0.5	OSPF	78	LS Acknowledge	10:19:21.005264
44	42.143324	fe80::200:ff:fea:6	ff02::5	OSPF	94	Hello Packet	10:19:21.111872
45	51.934916	10.0.3.2	224.0.0.5	OSPF	82	Hello Packet	10:19:30.903464
46	51.973802	fe80::200:ff:fea:7	ff02::5	OSPF	94	Hello Packet	10:19:30.942350
47	51.993551	10.0.3.1	224.0.0.5	OSPF	82	Hello Packet	10:19:30.962099
48	52.160497	fe80::200:ff:fea:6	ff02::5	OSPF	94	Hello Packet	10:19:31.129045
49	57.706462	10.0.3.2	224.0.0.5	OSPF	122	LS Update	10:19:36.675010

(a) Inexistência de tráfego no *router Ra*.

55	31.513101	10.0.9.10	10.0.7.20	ICMP	98	Echo (ping) repl	10:19:22.458151
56	32.512921	10.0.7.20	10.0.9.10	ICMP	98	Echo (ping) requ	10:19:23.457971
57	32.513177	10.0.9.10	10.0.7.20	ICMP	98	Echo (ping) repl	10:19:23.458227
58	33.512851	10.0.7.20	10.0.9.10	ICMP	98	Echo (ping) requ	10:19:24.457901
59	33.513128	10.0.9.10	10.0.7.20	ICMP	98	Echo (ping) repl	10:19:24.458178
60	34.512843	10.0.7.20	10.0.9.10	ICMP	98	Echo (ping) requ	10:19:25.457993
61	34.513097	10.0.9.10	10.0.7.20	ICMP	98	Echo (ping) repl	10:19:25.458147
62	35.512843	10.0.7.20	10.0.9.10	ICMP	98	Echo (ping) requ	10:19:26.457993
63	35.513102	10.0.9.10	10.0.7.20	ICMP	98	Echo (ping) repl	10:19:26.458152
64	36.512849	10.0.7.20	10.0.9.10	ICMP	98	Echo (ping) requ	10:19:27.457999
65	36.513111	10.0.9.10	10.0.7.20	ICMP	98	Echo (ping) repl	10:19:27.458161
66	37.512843	10.0.7.20	10.0.9.10	ICMP	98	Echo (ping) requ	10:19:28.457893

(b) Pacotes de "ida" e de "volta" passam por Rc.

Fig. 29: Pacotes a fluir depois do reajuste da rede (10:19:14h).

Posteriormente, resta voltar a normalizar as ligações, isto é, voltar a colocar *up* o *eth0* do *router Ra* (**ifconfig eth0 up**). Este comando foi realizado às **10:19:36** [30]. Do mesmo modo, por volta da mesma hora, volta a existir troca de pacotes OSPF [31], com o objetivo de ajustar as tabelas de encaminhamento dos diferentes *routers*. A transmissão dos pacotes passa a ser feita pelos trajetos originais, isto é, os pacotes de "ida" (10.0.7.2 -> 10.0.9.10) passam por Ra e os pacotes de "volta" (10.0.9.10 -> 10.0.7.20) passam por Rc [25]. De forma a certificar que tudo voltou ao normal basta acedermos às novas tabelas de endereçamento e perceber que estas encontram-se agora iguais às tabelas iniciais [??].

```
root@Ra:/tmp/pycore.59148/Ra.conf# date +%T && ifconfig eth0 up
10:19:36
```

Fig. 30: Reposta a interface eth0 do *router Ra* (**10:19:36h**).

49	57.706462	10.0.3.2	224.0.0.5	OSPF	122 LS Update	10:19:36,675010
50	57.709431	10.0.3.1	224.0.0.5	OSPF	186 LS Update	10:19:36,677979
51	57.709884	10.0.3.2	224.0.0.5	OSPF	94 LS Update	10:19:36,678432
52	58.001425	10.0.3.2	224.0.0.5	OSPF	118 LS Acknowledge	10:19:36,969973
53	58.050916	10.0.3.1	224.0.0.5	OSPF	78 LS Acknowledge	10:19:37,019464
54	59.451683	fe80::200:ff:fea:6	ff02::5	OSPF	246 LS Update	10:19:38,420231
55	61.935073	10.0.3.2	224.0.0.5	OSPF	82 Hello Packet	10:19:40,903621
56	61.977050	fe80::200:ff:fea:7	ff02::5	OSPF	94 Hello Packet	10:19:40,945598
57	61.977466	10.0.3.2	224.0.0.5	OSPF	122 LS Update	10:19:40,946014
58	61.978423	10.0.3.2	224.0.0.5	OSPF	122 LS Update	10:19:40,946971
59	61.978771	10.0.3.2	224.0.0.5	OSPF	154 LS Update	10:19:40,947319
60	61.978999	10.0.3.1	224.0.0.5	OSPF	154 LS Update	10:19:40,947547
61	61.993499	10.0.3.1	224.0.0.5	OSPF	82 Hello Packet	10:19:40,962047
62	62.005488	10.0.3.2	224.0.0.5	OSPF	98 LS Acknowledge	10:19:40,974036
63	62.054296	10.0.3.1	224.0.0.5	OSPF	78 LS Acknowledge	10:19:41,022844
64	62.133766	fe80::200:ff:fea:6	ff02::5	OSPF	94 Hello Packet	10:19:41,102314
65	62.177788	fe80::200:ff:fea:7	ff02::5	OSPF	130 LS Update	10:19:41,146336
66	62.278148	fe80::200:ff:fea:7	ff02::5	OSPF	150 LS Update	10:19:41,246696
67	62.379947	fe80::200:ff:fea:6	ff02::5	OSPF	150 LS Update	10:19:41,348495
68	62.451953	fe80::200:ff:fea:7	ff02::5	OSPF	150 LS Acknowledge	10:19:41,420501
69	64.352147	fe80::200:ff:fea:7	ff02::5	OSPF	174 LS Update	10:19:43,320695
70	65.178935	fe80::200:ff:fea:6	ff02::5	OSPF	130 LS Acknowledge	10:19:44,147483
71	66.982139	10.0.3.1	224.0.0.5	OSPF	122 LS Update	10:19:45,950687
72	67.012352	10.0.3.2	224.0.0.5	OSPF	78 LS Acknowledge	10:19:45,980900

(a) Troca de pacotes OSPF em Ra.

90	45.731308	10.0.1.2	224.0.0.5	OSPF	122 LS Update	10:19:36,676358
91	45.731609	10.0.1.1	224.0.0.5	OSPF	154 LS Update	10:19:36,676659
92	46.063361	10.0.1.1	224.0.0.5	OSPF	78 LS Acknowledge	10:19:37,008411
93	46.087657	10.0.1.2	224.0.0.5	OSPF	98 LS Acknowledge	10:19:37,032707

(b) Troca de pacotes OSPF em Rc.

Fig. 31: Troca de pacotes OSPF por forma a reestabelecer a conectividade inicial.

```
vcmd
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        0.0.0.0        255.255.255.0 U        0 0          0 eth0
10.0.1.0        0.0.0.0        255.255.255.0 U        0 0          0 eth1
10.0.2.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.3.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.6.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.7.0        0.0.0.0        255.255.255.0 U        0 0          0 eth2
10.0.8.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.9.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.10.0       10.0.1.2      255.255.255.0 UG       0 0          0 eth1
root@Rb:/tmp/pycore.40591/Rb.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask        Flags   MSS Window irtt Iface
10.0.0.0        0.0.0.0        255.255.255.0 U        0 0          0 eth0
10.0.1.0        0.0.0.0        255.255.255.0 U        0 0          0 eth1
10.0.2.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.3.0        10.0.0.1       255.255.255.0 UG       0 0          0 eth0
10.0.6.0        10.0.0.1       255.255.255.0 UG       0 0          0 eth0
10.0.7.0        0.0.0.0        255.255.255.0 U        0 0          0 eth2
10.0.8.0        10.0.1.2       255.255.255.0 UG       0 0          0 eth1
10.0.9.0        10.0.0.1       255.255.255.0 UG       0 0          0 eth0
10.0.10.0       10.0.0.1      255.255.255.0 UG       0 0          0 eth0
root@Rb:/tmp/pycore.40591/Rb.conf#
```

Fig. 32: Atualização da tabela de encaminhamento do Rb.

3.8 Exercício 2.2.c

Questão

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 S1 localizado no departamento D. Use o comando route delete para o efeito. Que implicações tem esta medida para os utilizadores da empresa que acedem aos servidores. Justifique.

Resposta

Ao remover a rota 0.0.0.0 o router fica sem saber o que fazer quando encontra pacotes que não fazem match com os elementos que tem na tabela de endereçamento. Como seria de esperar, os utilizadores que estão no mesmo departamento (ligados à mesma subnet) têm conhecimento da topologia local e, por isso, não sofrem em nada com esta situação. No entanto, relativamente aos utilizadores externos ao departamento e que pretendem fazer um acesso ao servidor, todos os pacotes acabam como perdidos. O que acontece é que o servidor os recebe mas, sem a rota por defeito, não consegue dar resposta.

Realização

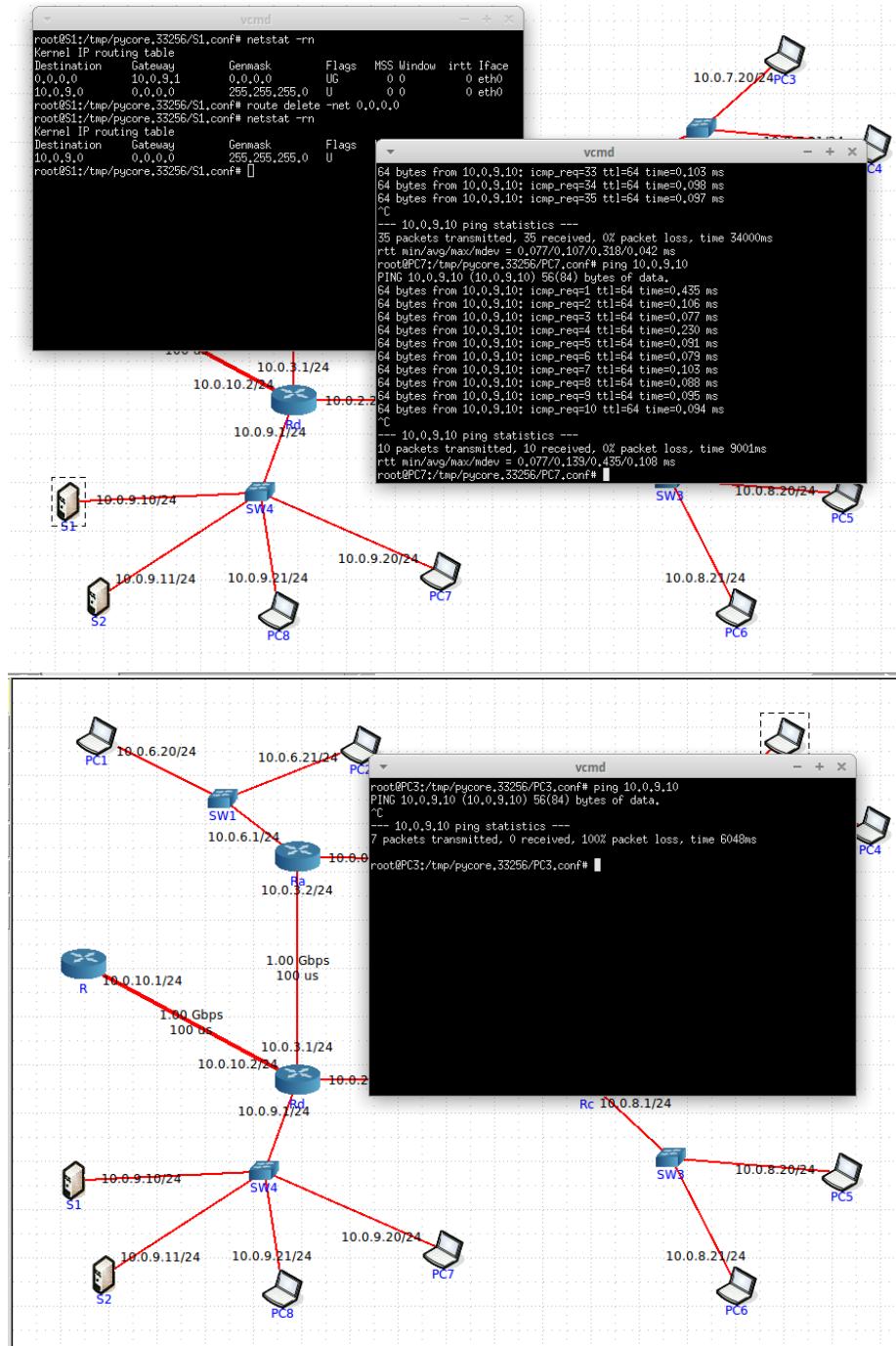


Fig. 33: Conexão com a rede local mantida, com o exterior perdido.

3.9 Exercício 2.2.d e 2.2.e

Questão

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.

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.

Resposta

Foram adicionadas as rotas correspondentes aos comandos: **route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.9.1**, **route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.9.1**, **route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.9.1**. Estas rotas foram adicionadas de forma a que a conexão entre departamentos fosse restabelecida, uma vez que se registou o destinatário de cada um e se implicou a resposta através do gateway do departamento Rd.

É possível ver o processo de retirada da *route 0.0.0.0* (default) e adição das estáticas correspondentes a cada um dos departamentos.

Realização

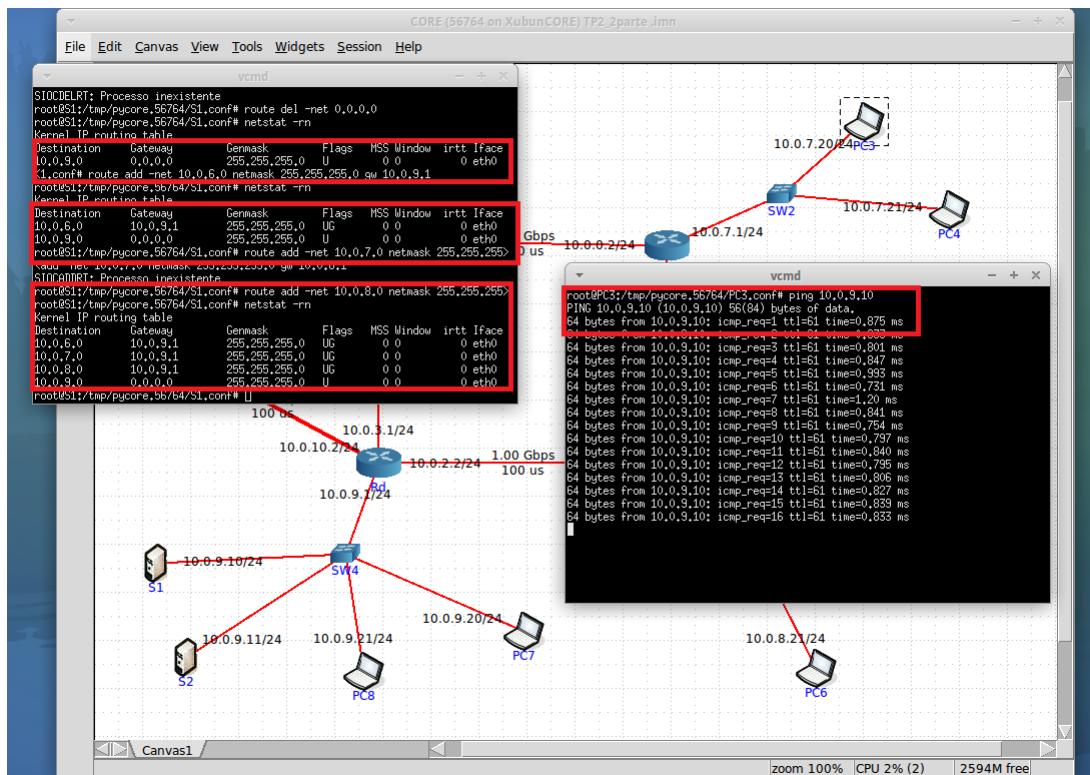


Fig. 34: A conexão é restabelecida com a nova tabela de reencaminhamento.

3.10 Exercício 3.1

Questão

Considere que dispõe apenas do endereço de rede IP 172.16.0.0/16, 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.

Resposta

A partir do endereço dado, decidiu-se criar um conjunto de subredes de endereço 172.16.0.0/24, dando 8 bits para criação de subredes e 8 bits para criação de *hosts* dentro de cada subrede, de modo a maximizar a flexibilidade da rede para instalações futuras. Prosseguimos então à redefinição do esquema de endereçamento:

- **Ra:** Definimos o IP como 172.16.1.1/24 na interface eth2 (ligação ao Departamento A). A escolha do terceiro octeto a 1 reside no facto de este ter sido a primeira subrede a ser endereçada e 1 no quarto octeto reside no facto de ser o primeiro dispositivo/host da subrede. Propagou-se então os endereços da subnetting para os PC's do departamento. No caso do PC1 recebeu o endereço IP 172.16.1.2/24 e o PC2 recebeu o endereço IP 172.16.1.3/24.
- **Rb:** Repetiu-se o processo mas com 172.16.2.1/24 para a interface eth2 do router (ligação ao Departamento B). A escolha do terceiro octeto a 2 reside no facto de este ter sido a segunda subrede a ser endereçada. Propagou-se o IP da subnetting para os PC3 e PC4 com os IP 172.16.2.2/24 e 172.16.2.3/24, respetivamente.
- **Rc:** Repetiu-se o processo mas com 172.16.3.1/24 para a interface eth2 do router (ligação ao Departamento C). A escolha do terceiro octeto a 3 reside no facto de este ter sido a terceira subrede a ser endereçada. Propagou-se o IP da subnetting para os PC5 e PC6 com os IP 172.16.3.2/24 e 172.16.3.3/24, respetivamente.
- **Rd:** Repetiu-se o processo mas com 172.16.4.1/24 para a interface eth4 do router (ligação ao Departamento D). A escolha do terceiro octeto a 4 reside no facto de este ter sido a quarta subrede a ser endereçada. Propagou-se para PC7, PC8, S1 e S2 172.16.4.2/24, 172.16.4.3/24, 172.16.4.4/24 e 172.16.4.5/24, respetivamente.

Realização

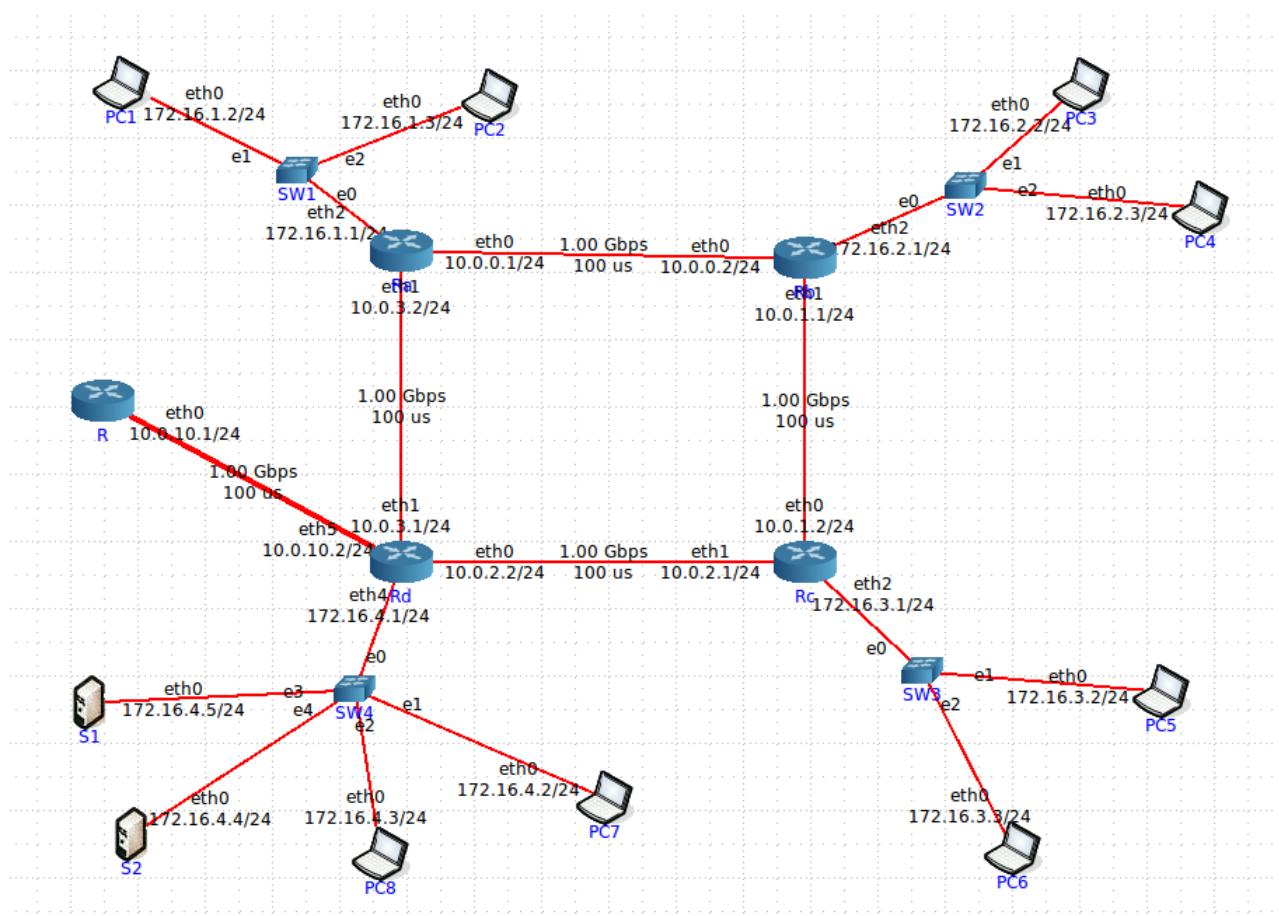


Fig. 35: As subredes mantêm a capacidade de enviar pacotes entre si.

3.11 Exercício 3.2

Questão

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

Resposta

A máscara da rede IP é 255.255.0.0 , sendo a máscara para cada subrede 255.255.255.0 . Isto permite-nos 8 bits para endereçar os *hosts* de cada departamento. Como os *hosts* 172.16.x.0/24 e 172.16.x.255/24 estão reservados como o identificador da subrede e acesso de *broadcast*, respetivamente, dispomos de $2^8 - 2 = 254$ *hosts* por departamento.

3.12 Exercício 3.3

Questão

Garanta e verifique que conectividade IP entre as várias redes locais da organização MIEI-RC é mantida.

Resposta

Recorrendo ao comando *ping*, verificou-se que a conectividade IP das subredes da organização e confirmou-se que cada subrede se pode ligar a cada uma das outras subredes locais.

Realização

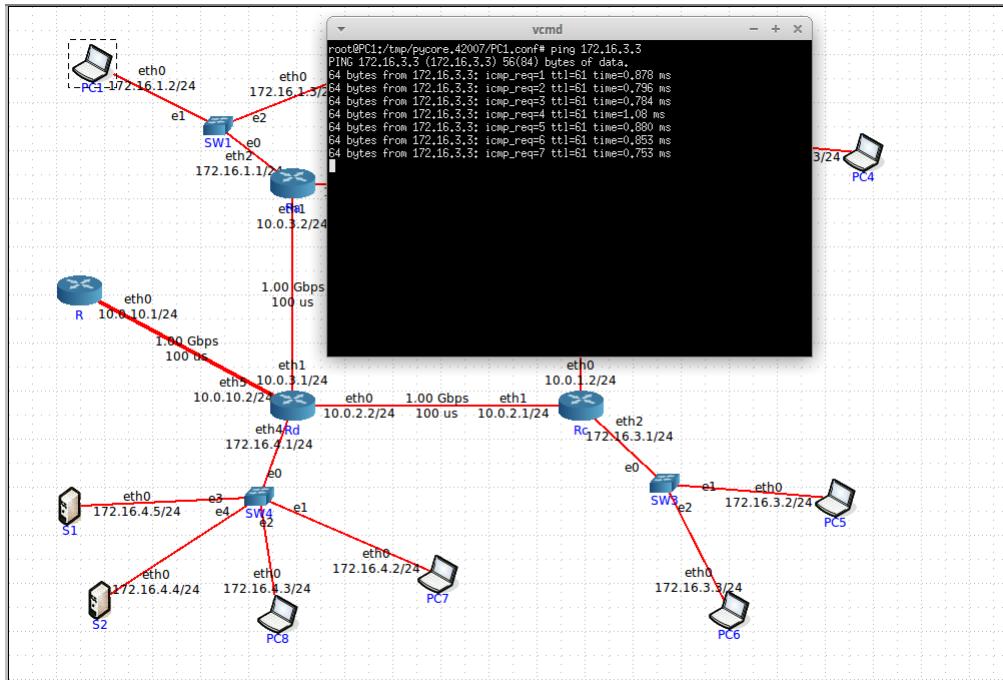


Fig. 36: As subredes mantêm a capacidade de enviar pacotes entre si.

4 Conclusions

Neste trabalho aprofundou-se a compreensão do funcionamento interno das redes IP, principalmente nas suas estruturas e na transmissão de dados através da mesma. Particularmente, reforçou-se o conhecimento e compreensão do *subnetting* e as suas aplicações. Consequentemente, conhecemos as potencialidades do *routing* dinâmico e estático, bem como a importância das tabelas de endereçamento. No que toca aos datagramas, tornou-se mais evidente a importância do TTL, dos diferentes protocolos IP e todo um outro conjunto de informações presentes tanto em *headers* como em *payloads*.

References

1. : Internet Control Message Protocol. RFC 792 (1981)
2. Wikipedia: Internet control message protocol. https://pt.wikipedia.org/wiki/Internet_Control_Message_Protocol (2017) [Online; acedido a 4-Novembro-2017].