

# 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

Temos que fazer...

## 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

### 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*.

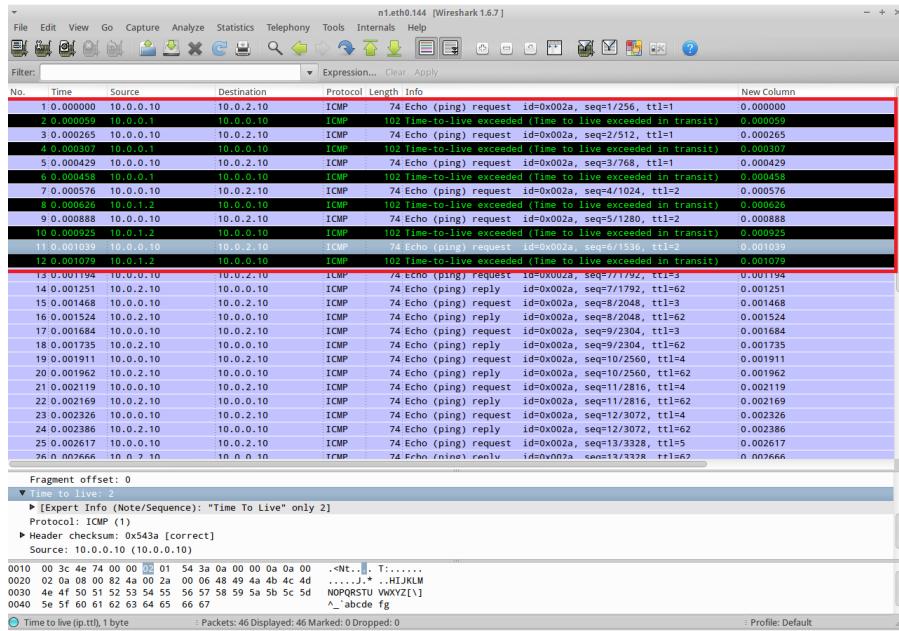


Fig. 1: TTL excedido.

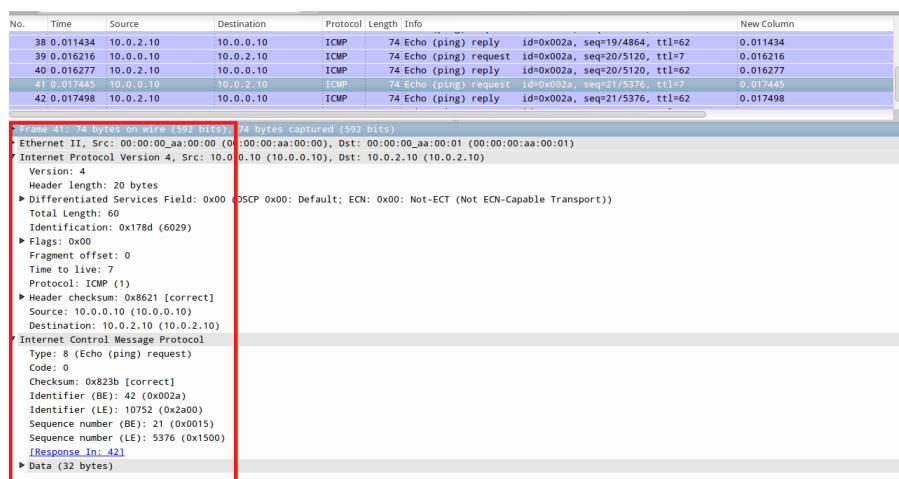


Fig. 2: Tamanho do pacote e divisão.

## Realização

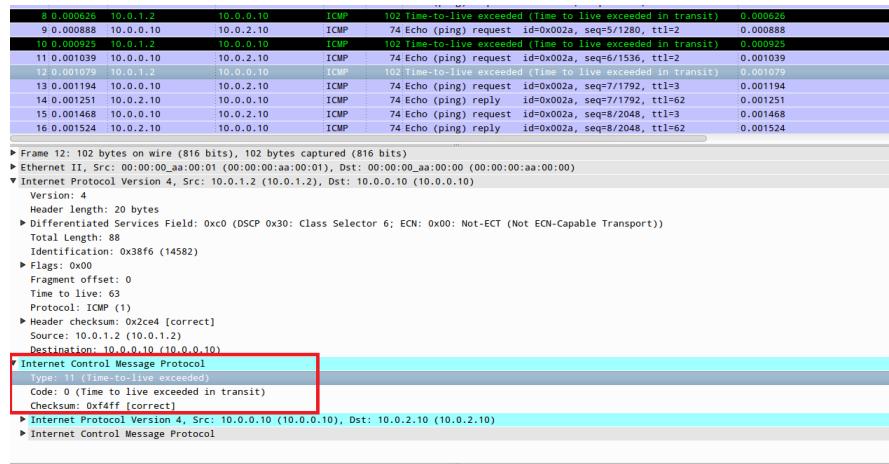


Fig. 3: Mensagem com ICMP retornando TLL excedido.

## 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

## 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

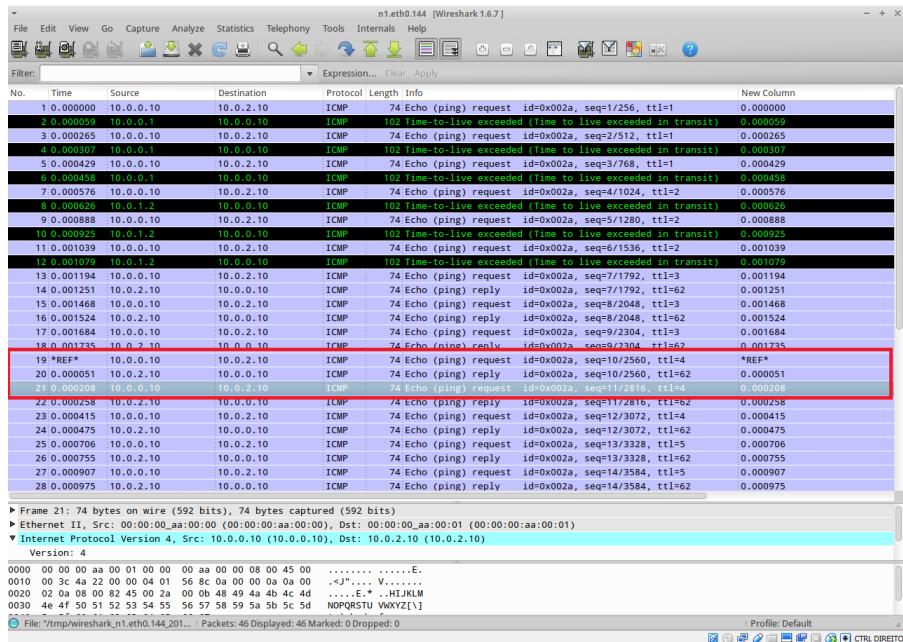


Fig. 4: Round-Trip Time.

No.	Time	Source	Destination	Protocol	Length	Info	New Column
1	0.000000	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=1/256, ttl=1	0.000000
2	0.000059	10.0.0.1	10.0.0.10	ICMP	102	Time to live exceeded (Time to live exceeded in transit)	0.000059
3	0.000265	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=2/512, ttl=1	0.000265
4	0.000307	10.0.0.1	10.0.0.10	ICMP	102	Time to live exceeded (Time to live exceeded in transit)	0.000307
5	0.000429	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=3/768, ttl=1	0.000429
6	0.000458	10.0.0.1	10.0.0.10	ICMP	102	Time to live exceeded (Time to live exceeded in transit)	0.000458
7	0.000576	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=4/1024, ttl=2	0.000576
8	0.000626	10.0.1.2	10.0.0.10	ICMP	102	Time to live exceeded (Time to live exceeded in transit)	0.000626
9	0.000888	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=5/1280, ttl=2	0.000888
10	0.000925	10.0.1.2	10.0.0.10	ICMP	102	Time to live exceeded (Time to live exceeded in transit)	0.000925
11	0.001039	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=6/1536, ttl=2	0.001039
12	0.001079	10.0.1.2	10.0.0.10	ICMP	102	Time to live exceeded (Time to live exceeded in transit)	0.001079
13	0.001194	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=7/1792, ttl=3	0.001194
14	0.001251	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=7/1792, ttl=62	0.001251
15	0.001468	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=8/2048, ttl=3	0.001468
16	0.001524	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=8/2048, ttl=62	0.001524
17	0.001684	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=9/2304, ttl=3	0.001684
18	0.001735	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=9/2304, ttl=62	0.001735
19	*REF*	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=10/2560, ttl=4	*REF*
20	0.000051	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=10/2560, ttl=62	0.000051
21	0.000208	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=11/2816, ttl=4	0.000208
22	0.000258	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=11/2816, ttl=62	0.000258
23	0.000415	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=12/3072, ttl=4	0.000415
24	0.000475	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=12/3072, ttl=62	0.000475
25	0.000706	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=13/3328, ttl=5	0.000706
26	0.000755	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=13/3328, ttl=62	0.000755
27	0.000997	10.0.0.10	10.0.2.10	ICMP	74	Echo (ping) request id=0x002a, seq=14/3584, ttl=5	0.000997
28	0.000975	10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x002a, seq=14/3584, ttl=62	0.000975

Fig. 5: Identificação do endereço IP.

## 2.5 Exercício 2.b.

### Questão

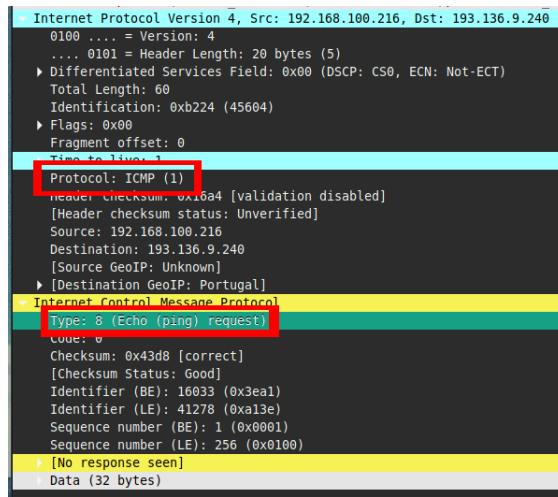
Qual é o valor do campo protocolo? O que identifica?

### 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



The screenshot shows the NetworkMiner tool interface with an analyzed ICMP echo request packet. The packet details are as follows:

- Internet Protocol Version 4:** Src: 192.168.100.216, Dst: 193.136.9.240
- Protocol:** ICMP (1) [highlighted by a red box]
- Type:** 8 (Echo (ping) request) [highlighted by a red box]
- Code:** 0
- Checksum:** 0x43d8 [correct]
- Identifier (BE):** 16033 (0x3e1)
- Identifier (LE):** 41278 (0xa13e)
- Sequence number (BE):** 1 (0x0001)
- Sequence number (LE):** 256 (0x0100)
- Data:** (32 bytes) [No response seen]

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

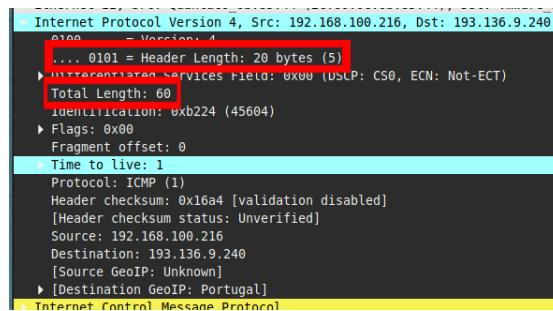


Fig. 7: Identificação do tamanho do *header* e do *payload*.

## 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

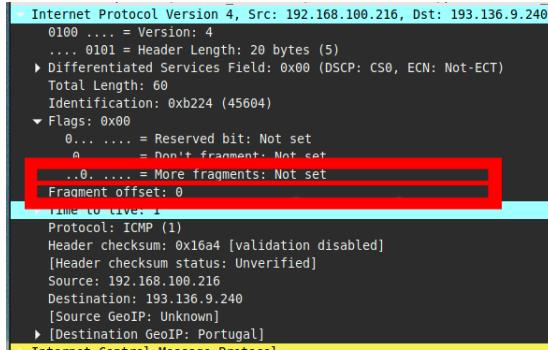


Fig. 8: Fragmentação.

## 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 *identificação* 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

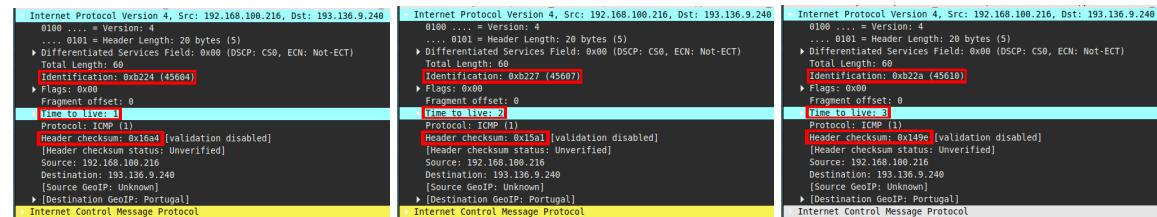


Fig. 9: Campos que mudam.

## 2.9 Exercício 2.f.

### Questão

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

### Resposta

O campo da identificação 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.

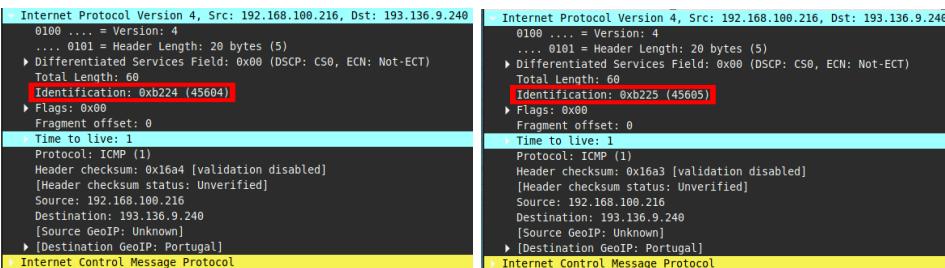


Fig. 10: Identificação.

## 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 utilizar 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 *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

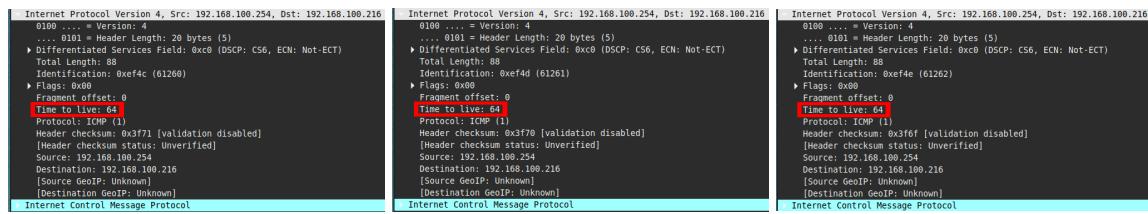


Fig. 11: Router de acesso com *TTL* = 64.

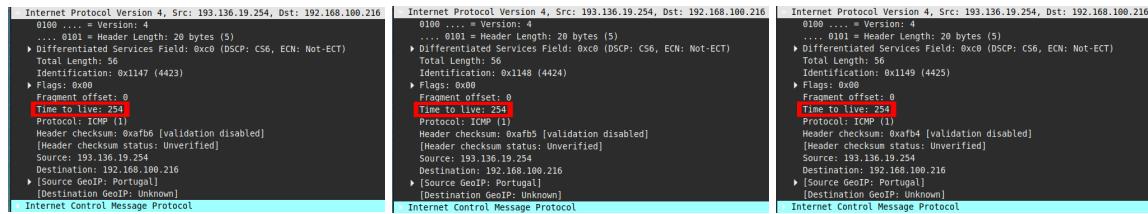


Fig. 12: Segundo router com *TTL* = 254.

## 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	9.318609576	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xa8ca A d1-debug.dropbox.com
12	9.318609541	192.168.100.216	192.168.100.254	DNS	80	Standard query 0xbcd4 AAAA d1-debug.dropbox.com
13	9.312548779	192.168.100.216	192.168.100.216	DNS	190	Standard query response 0xbcd4 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	9.649437033	192.168.100.216	192.168.100.254	DNS	347	Standard query response 0x3dcf A marco.uminho.pt A 193.136.0.240 ID=6 dns3.uminho.
16	9.650136240	192.168.100.254	192.168.100.216	DNS	340	Standard query 0x3dcf A marco.uminho.pt A 193.136.0.240 ID=6 dns3.uminho.
17	9.650136277	192.168.100.216	192.168.100.216	DNS	340	Standard query 0x3dcf A marco.uminho.pt A 193.136.0.240 ID=6 dns3.uminho.
18	9.656880505	192.168.100.216	193.136.0.240	IPv4	1514	1514 Fragmented IP protocol [proto=ICMP 1, off=1480, ID=811] [Reassembled in #20]
19	9.656880397	192.168.100.216	193.136.0.240	IPv4	1514	1514 Fragmented IP protocol [proto=ICMP 1, off=1480, ID=811] [Reassembled in #20]
*	20	9.656880359	192.168.100.216	193.136.0.240	ICMP	87 Echo (ping) request id=0xa4f3, seq=1/256, ttl=1 (no response found)
*	21	9.656880461	192.168.100.216	193.136.0.240	IPv4	1514 Fragmented IP protocol [proto=ICMP 1, off=0, ID=814] [Reassembled in #23]
*	22	9.656897423	192.168.100.216	193.136.0.240	IPv4	1514 Fragmented IP protocol [proto=ICMP 1, off=1480, ID=814] [Reassembled in #23]
*	23	9.656890289	192.168.100.216	193.136.0.240	IPv4	87 Echo (ping) request id=0xa4f3, seq=2/252, ttl=1 (no response found)
*	24	9.656890290	192.168.100.216	193.136.0.240	IPv4	1514 Fragmented IP protocol [proto=ICMP 1, off=1480, ID=815] [Reassembled in #26]
*	25	9.656904680	192.168.100.216	193.136.0.240	IPv4	1514 Fragmented IP protocol [proto=ICMP 1, off=1480, ID=815] [Reassembled in #26]
*	26	9.656905558	192.168.100.216	193.136.0.240	ICMP	87 Echo (ping) request id=0xa4f3, seq=3/256, ttl=1 (no response found)
*	27	9.656909521	192.168.100.216	193.136.0.240	IPv4	1514 Fragmented IP protocol [proto=ICMP 1, off=0, ID=816] [Reassembled in #20]
*	28	9.656910423	192.168.100.216	193.136.0.240	IPv4	1514 Fragmented IP protocol [proto=ICMP 1, off=1480, ID=816] [Reassembled in #20]
*	29	9.656911383	192.168.100.216	193.136.0.240	ICMP	87 Echo (ping) request id=0xa4f3, 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_0:b8:54 (2c:60:0:c6:b8:54), Dst: Vmware_d2:19:f0 (00:0c:29:d2:19:f0)						
> Internet Protocol Version 4, Src: 192.168.100.216, Dst: 193.136.0.240						
0x0000 Version: 4, Header Length: 20 bytes (5)						
0x0004 Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)						
Total Length: 1500						
Identification: 0x8a13 (43827)						
Flags: 0x01 (More Fragments)						
0... = More Fragments: Not set						
..0... = Don't Fragment: Not set						
..1.... = More fragments: Set						
Fragment offset: 0						
Time to live: 1						
Protocol: ICMP (1)						
Checksum: 0x0f84 (Validation disabled)						
[Header checksum status: Unverified]						
Source: 192.168.100.216						
Destination: 193.136.0.240						
[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

## 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?

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	9.319254079	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 response 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 193.136.9.244 NS dns3.uminho.pt
17	9.650723657	192.168.100.254	192.168.100.216	DNS	129	Standard query response 0x78d8 AAAA marco.uminho.pt SOA dns3.uminho.pt
18	9.656885985	192.168.100.216	193.136.9.248	ICMP	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a13) [Reassembled in #20]
*	18.9.656885985	192.168.100.216	193.136.9.248	ICMP	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a13) [Reassembled in #20]
*	18.9.656885985	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request 1d=0x4af3, seq=1/256, ttl=1 (no response found!)
21	9.656896401	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a14) [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=8a14) [Reassembled in #23]
23	9.656898295	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request 1d=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=8a15) [Reassembled in #26]
25	9.656904600	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a15) [Reassembled in #26]
26	9.656905211	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a15) [Reassembled in #29]
27	9.656909521	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a15) [Reassembled in #29]
28	9.656918428	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, Id=8a15) [Reassembled in #29]
29	9.656911383	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request 1d=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: 1580						
*	Identification: 0xa8a13 (48027)					
*	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: 0xfbf4 [validation disabled]					
>	[Header checksum status: Unverified]					
>	Source: 192.168.100.216					
>	Destination: 193.136.9.248					
>	[Source GeoIP: Unknown]					
>	[Destination GeoIP: Unknown]					
>	Reassembled TPDU in frame: 28					
>	Data (1480 bytes)					

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

## 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 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	9.319254079	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 response 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 193.136.9.244 NS dns3.uminho.pt
17	9.650723657	192.168.100.254	192.168.100.216	DNS	129	Standard query response 0x78d8 AAAA marco.uminho.pt SOA dns3.uminho.pt
18	9.656885985	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a13) [Reassembled in #20]
*	18.9.656885985	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a13) [Reassembled in #20]
*	18.9.656885985	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request 1d=0x4af3, seq=1/256, ttl=1 (no response found!)
21	9.656896401	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a14) [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=8a14) [Reassembled in #23]
23	9.656898295	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request 1d=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=8a15) [Reassembled in #26]
25	9.656904600	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=0, Id=8a15) [Reassembled in #26]
26	9.656905558	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request 1d=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=8a16) [Reassembled in #29]
28	9.656918428	192.168.100.216	193.136.9.248	IPV4	1514	Fragmented IP protocol (proto=ICMP 1, off=1480, Id=8a16) [Reassembled in #29]
29	9.656911383	192.168.100.216	193.136.9.248	ICMP	87	Echo (ping) request 1d=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: 1580						
*	Identification: 0xa8a13 (48027)					
*	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: 0xfbf4 [validation disabled]					
>	[Header checksum status: Unverified]					
>	Source: 192.168.100.216					
>	Destination: 193.136.9.248					
>	[Source GeoIP: Unknown]					
>	[Destination GeoIP: Unknown]					
>	Reassembled TPDU in frame: 28					
>	Data (1480 bytes)					

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 datagramas capturados pelo Wireshark, percebe-se que existem três datagramas 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.

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

### Realização

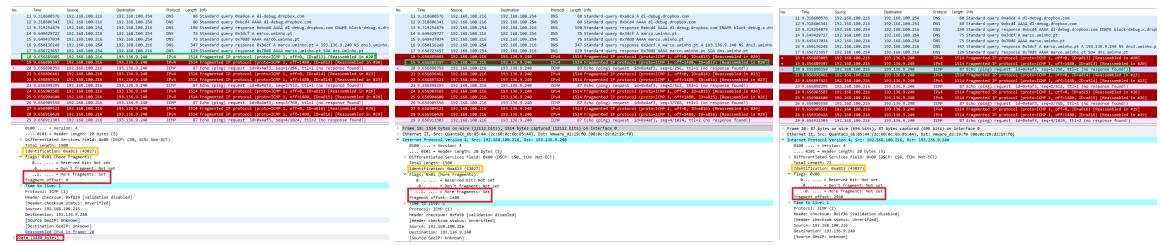


Fig. 16: Encontrando vários fragmentos com a mesma *identification*, é possível saber a sua ordem através do *fragment offset* e flags.

## 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 ( $n = 1, 2, \dots$ ).

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.  
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.

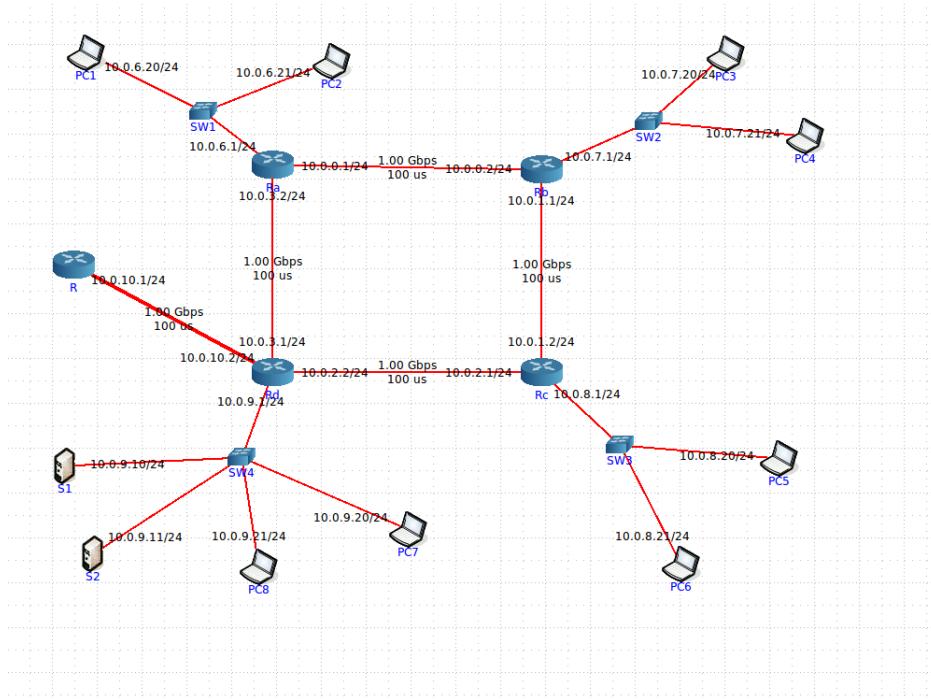


Fig. 17: Topologia da rede.

#### 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

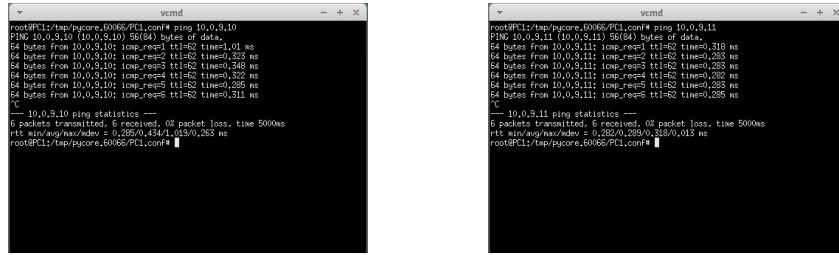


Fig. 18: Teste de conectividade entre PC1 e os servidores S1 (à esquerda) e S2 (à direita).

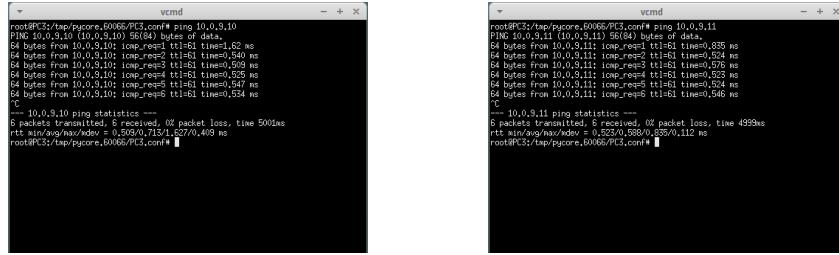


Fig. 19: Teste de conectividade entre PC3 e os servidores S1 (à esquerda) e S2 (à direita).

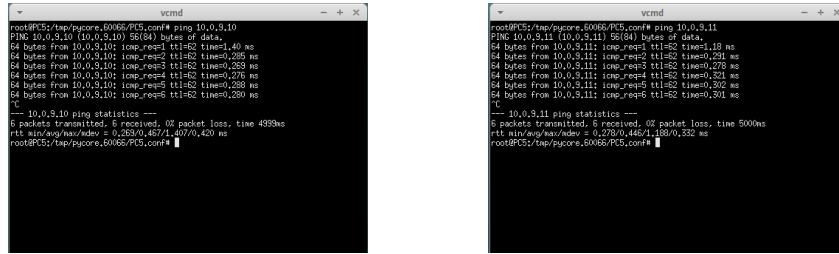


Fig. 20: Teste de conectividade entre PC5 e os servidores S1 (à esquerda) e S2 (à direita).

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

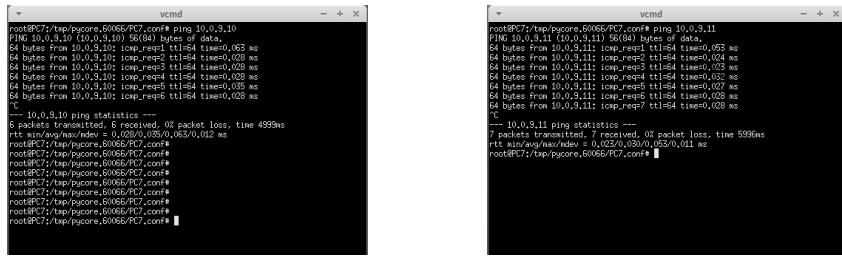


Fig. 21: Teste de conectividade entre PC7 e os servidores S1 (à esquerda) e S2 (à direita).

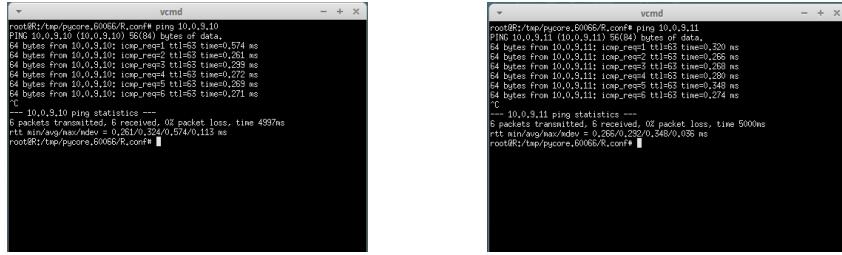


Fig. 22: Teste de conectividade entre R e os servidores S1 (à esquerda) e S2 (à direita).

Tendo em conta os resultados obtidos em 23, 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 ).

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 incluí 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 figure consists of two vertically stacked terminal windows, both titled "vcmd".

**Terminal 1 (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  UG    0 0          0 eth0
root@PC3:/tmp/pycore_60066/PC3.conf#
```

**Terminal 2 (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  UG    0 0          0 eth0
10.0.1.0         0.0.0.0        255.255.255.0  UG    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  UG    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

### Resposta

## Realização

### **3.8 Exercício 2.2.c**

**Questão**

**Resposta**

**Realização**

### **3.9 Exercício 2.2.d**

**Questão**

**Resposta**

**Realização**

### **3.10 Exercício 2.2.e**

**Questão**

**Resposta**

**Realização**

### **3.11 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. 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 do router. 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 do router. 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 do router. 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**

#### **3.12 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.

### **Realização**

#### **3.13 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.

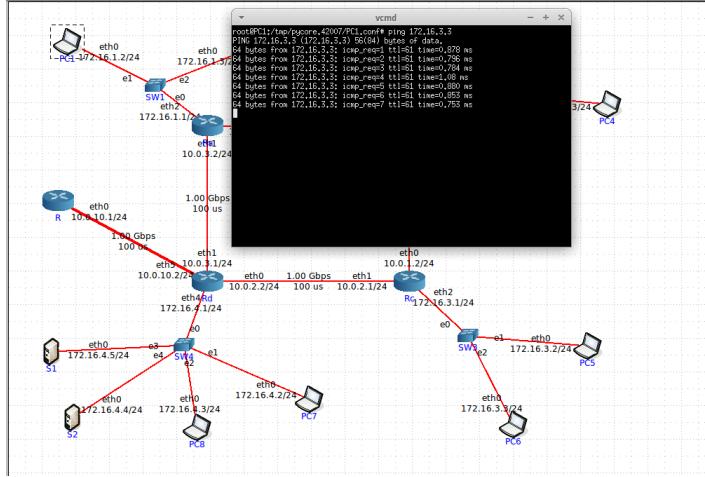


Fig. 24: As subredes mantêm a capacidade de enviar pacotes entre si.

## Realização

## 4 Conclusions

Neste trabalho...

## References

1. : Internet Control Message Protocol. RFC 792 (1981)
2. Wikipedia: Internet control message protocol. [https://pt.wikipedia.org/wiki/Internet\\_Control\\_Message\\_Protocol](https://pt.wikipedia.org/wiki/Internet_Control_Message_Protocol) (2017) [Online; accessed on 4-November-2017].