

Universidade do Minho

REDES DE COMPUTADORES

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA

DATAGRAMAS, FRAGMENTAÇÃO, ENDEREÇAMENTO E ENCAMINHAMENTO IPv4

[TP2]

GRUPO 9

A85227 João Pedro Rodrigues Azevedo

A85729 Paulo Jorge da Silva Araújo

A83719 Pedro Filipe Costa Machado

Braga
Novembro 2019

Conteúdo

1	Introdução	2
2	Parte I	3
2.1	Questões e Respostas	3
3	Parte II	12
3.1	Questões e Respostas	12
4	Conclusões	20

Capítulo 1

Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular de Redes de computadores, trata-se do segundo trabalho prático e teve como objetivo principal o estudo de Datagramas IP, Fragmentação de pacotes, endereçamento de componentes de uma rede e encaminhamento.

Na primeira parte deste estudo é realizado o registo de datagramas IP enviados e recebidos através da execução do programa traceroute. São analisados os vários campos de um datagrama IP e detalhado o processo de fragmentação realizado pelo IP.

Na segunda parte serão estudadas algumas das técnicas mais relevantes que foram propostas para aumentar a escalabilidade do protocolo IP, mitigar a exaustão dos endereços IPv4 e também reduzir os recursos de memória necessários nos routers para manter as tabelas de encaminhamento.

Assim, ao longo deste relatório vão ser apresentadas as várias questões enunciadas e respostas estruturadas às mesmas com diversas demonstrações práticas da sua validade levando a uma secção final de conclusões onde fazemos um balanço de todo o trabalho realizado e a sua importância nesta Unidade Curricular.

Capítulo 2

Parte I

2.1 Questões e Respostas

Pergunta 1

”Prepare uma topologia no *CORE* para verificar o comportamento do *traceroute*. Ligue um host (servidor) *s1* a um router *r2*; o router *r2* a um router *r3*, o router *r3* a um router *r4*, que por sua vez, se liga a um host (pc) *h5*. (Note que pode não existir conectividade IP imediata entre *s1* e *h5* até que o routing estabilize). Ajuste o nome dos equipamentos atribuídos por defeito para a topologia do enunciado.”

Análise do enunciado:

No primeiro exercício era pedido para criar uma topologia de rede no software emulador *CORE* para registrar tráfego enviado pelo comando *traceroute*.

A topologia de rede estabelecida foi a seguinte:

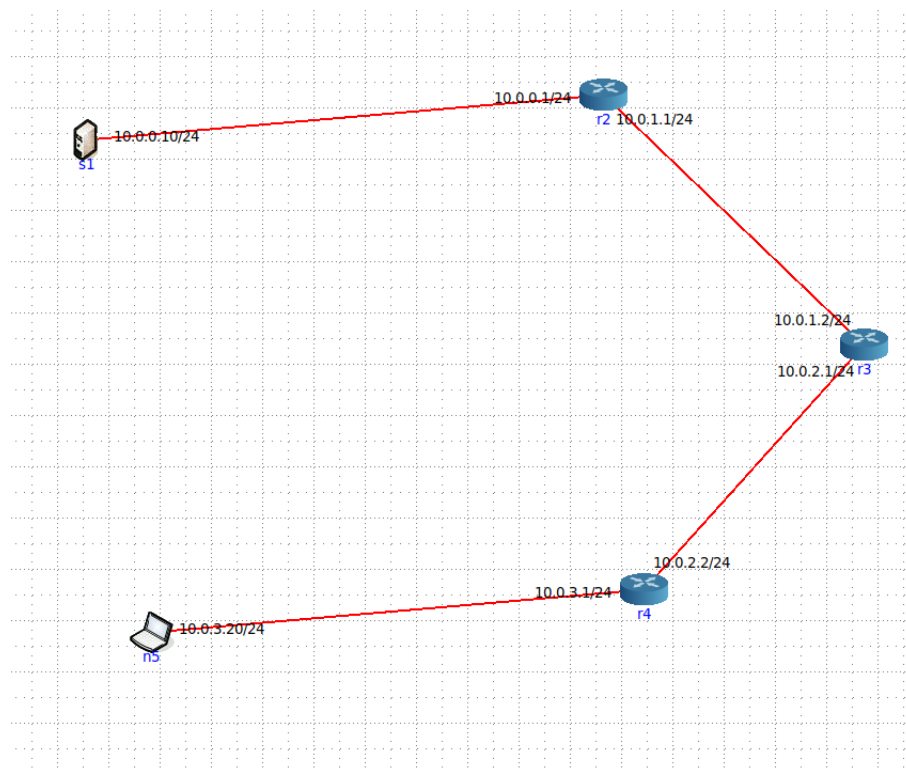


Figura 2.1: Topologia de rede no software *CORE*.

a) "Active o wireshark ou o tcpdump no pc s1. Numa shell de s1, execute o comando *tracert* -I para o endereço IP do host h5."

R: Como fase inicial era necessário ligar o Wireshark em s1 de modo a registar o tráfego gerado pelo seguinte comando:

```
$tracert -I 10.0.3.20
```

onde 10.0.3.20 representa o endereço IPv4 do host h5 (endereço destino do pacote).

b) "Registe e analise o tráfego ICMP enviado por s1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado."

R: O tráfego ICMP obtido no Wireshark foi o seguinte:

No.	Time	Source	Destination	Protocol	Length	Info
23	39.232279239	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=1/256, ttl=1 (no response found!)
24	39.232289996	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
25	39.232296323	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=2/512, ttl=1 (no response found!)
26	39.232299983	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
27	39.232302817	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=3/768, ttl=1 (no response found!)
28	39.232305558	10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
29	39.232308897	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=4/1024, ttl=2 (no response found!)
30	39.232321638	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
31	39.232324528	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=5/1280, ttl=2 (no response found!)
32	39.232329581	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
33	39.232332311	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=6/1536, ttl=2 (no response found!)
34	39.232336720	10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
35	39.232339876	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=7/1792, ttl=3 (no response found!)
36	39.232353813	10.0.2.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
37	39.232356557	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=8/2048, ttl=3 (no response found!)
38	39.232362891	10.0.2.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
39	39.232365423	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=9/2304, ttl=3 (no response found!)
40	39.232371514	10.0.2.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
41	39.232374707	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=10/2560, ttl=4 (reply in 42)
42	39.232397748	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x001e, seq=10/2560, ttl=61 (request in 41)
43	39.232401457	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=11/2816, ttl=4 (reply in 44)
44	39.232409776	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x001e, seq=11/2816, ttl=61 (request in 43)
45	39.232412447	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=12/3072, ttl=4 (reply in 46)
46	39.232420687	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x001e, seq=12/3072, ttl=61 (request in 45)
47	39.232423962	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=13/3328, ttl=5 (reply in 48)
48	39.232431845	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x001e, seq=13/3328, ttl=61 (request in 47)
49	39.232434477	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=14/3584, ttl=5 (reply in 50)
50	39.232442163	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x001e, seq=14/3584, ttl=61 (request in 49)
51	39.232444689	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=15/3840, ttl=5 (reply in 52)
52	39.232452351	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x001e, seq=15/3840, ttl=61 (request in 51)
53	39.232455449	10.0.0.10	10.0.3.20	ICMP	74	Echo (ping) request id=0x001e, seq=16/4096, ttl=6 (reply in 54)
54	39.232463204	10.0.3.20	10.0.0.10	ICMP	74	Echo (ping) reply id=0x001e, seq=16/4096, ttl=61 (request in 53)

Figura 2.2: Tráfego ICMP registado pela execução do comando *tracert*.

Para obter apenas este tráfego foi necessário aplicar o filtro "icmp" na aba acima de cor verde.

Partindo então desta captura conseguimos analisar o seguinte:

O programa *tracert* enviou 10 pacotes "echo (ping) request", sendo o registo 41 o último a partir do qual, no seguinte, registo 42, obteve resposta por parte do destino.

O que se verifica, de forma clara, a partir do tráfego capturado, são as constantes mensagens de "Time-to-live exceeded (...)" obtidas antes da resposta positiva verificada no registo 42.

Partindo do princípio que o comando *tracert* envia pacotes aumentando gradualmente o TTL temos que, só a partir da décima mensagem, com TTL = 4, obtivemos resposta, o que significa que só fazendo 4 hops (saltos) se chega ao destino e assim, comparando com a topologia de rede, os resultados são o que esperávamos visto que temos 3 routers e um host até chegar ao destino.

O tráfego ICMP recebido como resposta sob a forma "echo (ping) reply" a partir do registo 42 serve como confirmação de como o número de hops é suficiente.

Nota: As mensagens seguintes repetidas (*request* e *reply*) são utilizadas pelo *tracert* por questões de cálculo estatístico de "Round-Trip Time".

c) "Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino h5? Verifique na prática que a sua resposta está correta."

R: Pelas razões apresentadas acima, o valor de TTL para alcançar o destino h5 deve ser de

TTL = 4.

Um primeiro sinal de que, na prática, TTL = 4 seria suficiente, seria o facto de obtermos respostas apenas quando o TTL definido nos "requests" é de 4 ou mais, como se pode ver na Figura 2.4.

41 *REP*	10.0.0.10	10.0.3.20	ICMP	74 Echo (ping) request	id=0x001e, seq=10/2560, ttl=4 (reply in 42)
42 0.000023041	10.0.3.20	10.0.0.10	ICMP	74 Echo (ping) reply	id=0x001e, seq=10/2560, ttl=61 (request in 41)

Figura 2.3: Caption

Por outro lado, se estabelecermos um TTL máximo menor que 4, por exemplo, de 3, conseguimos verificar que o pacote não passa do *gateway* 10.0.2.2¹:

```
$tracert -I 10.0.3.20 -m 3
```

```
root@sl:/tmp/pycore.42081/s1.conf# tracert -I 10.0.3.20 -m 3
tracert to 10.0.3.20 (10.0.3.20), 3 hops max, 60 byte packets
h 1 gateway (10.0.0.1) 0.034 ms 0.006 ms 0.004 ms
h 2 10.0.1.2 (10.0.1.2) 0.015 ms 0.017 ms 0.006 ms
h 3 10.0.2.2 (10.0.2.2) 0.015 ms 0.008 ms 0.008 ms
root@sl:/tmp/pycore.42081/s1.conf# tracert -I 10.0.3.20
tracert to 10.0.3.20 (10.0.3.20), 30 hops max, 60 byte packets
h 1 gateway (10.0.0.1) 0.037 ms 0.005 ms 0.004 ms
h 2 10.0.1.2 (10.0.1.2) 0.013 ms 0.005 ms 0.005 ms
h 3 10.0.2.2 (10.0.2.2) 0.023 ms 0.007 ms 0.007 ms
h 4 10.0.3.20 (10.0.3.20) 0.016 ms 0.009 ms 0.009 ms
root@sl:/tmp/pycore.42081/s1.conf#
```

Figura 2.4: Caption

e, deste último, obteríamos apenas mensagens de "ICMP TTL exceeded messages".

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

R: O "Round-Trip Time", como o próprio enunciado o diz, traduz-se no tempo médio de ida-e-volta do pacote, ou seja, a média da diferença de tempo entre um pedido "echo (ping) request" e uma resposta "echo (ping) reply".

O cálculo do *RTT* pode ser feito por duas abordagens diferentes²: Usando o *Wireshark* e nele estabelecer referências de tempo entre um *request* e um *reply* com "Set/Unset time reference"; ou, de forma mais simples, adotar o resultado fornecido pela execução do *tracert*, sendo que, na última linha do *output* tempos esse tempo em três testes diferentes feitos pelo programa, como se pode ver a seguir:

```
root@sl:/tmp/pycore.41261/s1.conf# tracert -I 10.0.3.20
tracert to 10.0.3.20 (10.0.3.20), 30 hops max, 60 byte packets
h 1 gateway (10.0.0.1) 0.030 ms 0.005 ms 0.004 ms
h 2 10.0.1.2 (10.0.1.2) 0.014 ms 0.006 ms 0.045 ms
h 3 10.0.2.2 (10.0.2.2) 0.014 ms 0.008 ms 0.048 ms
h 4 10.0.3.20 (10.0.3.20) 0.016 ms 0.050 ms 0.011 ms
root@sl:/tmp/pycore.41261/s1.conf#
```

Figura 2.5: Execução de *tracert* para obtenção de um *RTT*.

Assim podemos verificar que o tempo médio será de:

$$RTT(avg) = \frac{0.016 + 0.050 + 0.011}{3} \approx 0.026ms \quad (2.1)$$

¹Interface associada a r4.

²Não será referido, mas através do comando *ping* também é possível estimar o tempo médio *RTT*.

Pergunta 2

”Pretende-se agora usar o traceroute na sua máquina nativa, e gerar de datagramas IP de diferentes tamanhos. (...) Selecione a primeira mensagem ICMP capturada (referente a (i) tamanho por defeito) e centre a análise no nível protocolar IP.”

Análise do enunciado:

Este exercício pedia que se registasse tráfego IP na nossa máquina nativa gerando datagramas de diferentes tamanhos. O primeiro passo foi capturar, usando o Wireshark, o tráfego gerado pelo traceroute para diferentes tamanhos de pacote:

- (i) Sem especificar (tamanho por defeito):

```
$traceroute -I marco.uminho.pt
```

- (ii) 42xx bytes, i.e, Grupo 09, então 4209 bytes:

```
$traceroute -I marco.uminho.pt 4209
```

Os outputs dos comandos inseridos foram os seguintes:

```
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 60 byte packets
 1 _gateway (192.168.2.1) 0.483 ms 0.482 ms 0.471 ms
 2 cisco.di.uminho.pt (193.136.19.254) 1.164 ms 1.516 ms 1.731 ms
 3 marco.uminho.pt (193.136.9.240) 1.164 ms 1.195 ms 1.224 ms
joao@azevedo-n550jk:~$
```

Figura 2.6: *traceroute* com tamanho default (pacotes de 60 bytes).

```
joao@azevedo-n550jk:~$ sudo traceroute -I marco.uminho.pt 4209
traceroute to marco.uminho.pt (193.136.9.240), 30 hops max, 4209 byte packets
 1 _gateway (192.168.2.1) 1.033 ms 1.271 ms 1.619 ms
 2 cisco.di.uminho.pt (193.136.19.254) 3.009 ms 3.352 ms 4.034 ms
 3 marco.uminho.pt (193.136.9.240) 4.962 ms 5.292 ms 5.645 ms
joao@azevedo-n550jk:~$
```

Figura 2.7: *traceroute* com tamanho definido (pacotes de 4209 bytes).

De seguida paramos a captura e concentramos a análise no tráfego ICMP gerado por estes comandos, tráfego esse registado pelo Wireshark.

Após filtrar os pacotes de modo a apresentar apenas os que tinham protocolo ICMP conseguimos identificar os diferentes ”echo (ping) request” e ”echo (ping) reply” dividindo os registos em 2 grupos.

O intervalo de registos 6 a 45 é referente ao primeiro comando (i) pois todas as mensagens possuem um tamanho igual de 74 bytes (60 bytes de pacote + 14 bytes ethernet spec.), como se pode ver na figura 2.8³.

³De notar que faltam alguns registos que não estão incluídos no *printscreen* para registos superiores a 72.

No.	Time	Source	Destination	Protocol	Length	Info
6	6.072424202	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=1/256, ttl=1 (no response found!)
7	6.072433026	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=2/512, ttl=1 (no response found!)
8	6.072435666	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=3/768, ttl=1 (no response found!)
9	6.072439133	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=4/1024, ttl=2 (no response found!)
10	6.072441530	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=5/1280, ttl=2 (no response found!)
11	6.072444681	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=6/1536, ttl=2 (no response found!)
12	6.072447708	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=7/1792, ttl=3 (reply in 29)
13	6.072450509	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=8/2048, ttl=3 (reply in 31)
14	6.072453039	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=9/2304, ttl=3 (reply in 32)
15	6.072456772	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=10/2560, ttl=4 (reply in 34)
16	6.072459588	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=11/2816, ttl=4 (reply in 35)
17	6.072462215	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=12/3072, ttl=4 (reply in 36)
18	6.072465613	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=13/3328, ttl=5 (reply in 37)
19	6.072469426	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=14/3584, ttl=5 (reply in 38)
20	6.072471349	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=15/3840, ttl=5 (reply in 39)
21	6.072473709	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=16/4096, ttl=6 (reply in 40)
22	6.073003759	192.168.2.1	192.168.2.151	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
23	6.073027564	192.168.2.1	192.168.2.151	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
24	6.073048471	192.168.2.1	192.168.2.151	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
25	6.073301619	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=17/4352, ttl=6 (reply in 42)
26	6.073308404	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=18/4608, ttl=6 (reply in 43)
27	6.073326866	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=19/4864, ttl=7 (reply in 44)
28	6.073863991	193.136.19.254	192.168.2.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
29	6.073975515	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=7/1792, ttl=62 (request in 12)
30	6.074067887	192.168.2.151	193.136.9.240	ICMP	74	Echo (ping) request id=0x12bf, seq=20/5120, ttl=7 (reply in 45)
31	6.074078996	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=8/2048, ttl=62 (request in 13)
32	6.074084170	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=9/2304, ttl=62 (request in 14)
33	6.074086477	193.136.19.254	192.168.2.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
34	6.074087869	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=10/2560, ttl=62 (request in 15)
35	6.074230767	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=11/2816, ttl=62 (request in 16)
36	6.074234423	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=12/3072, ttl=62 (request in 17)
37	6.074236816	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=13/3328, ttl=62 (request in 18)
38	6.074246183	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=14/3584, ttl=62 (request in 19)
39	6.074323439	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=15/3840, ttl=62 (request in 20)
40	6.074326577	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=16/4096, ttl=62 (request in 21)
41	6.074328210	193.136.19.254	192.168.2.151	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
42	6.074329631	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=17/4352, ttl=62 (request in 25)
43	6.074483945	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=18/4608, ttl=62 (request in 26)
44	6.074684781	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=19/4864, ttl=62 (request in 27)
45	6.074774261	193.136.9.240	192.168.2.151	ICMP	74	Echo (ping) reply id=0x12bf, seq=20/5120, ttl=62 (request in 30)
69	40.768376098	192.168.2.151	193.136.9.240	ICMP	1263	Echo (ping) request id=0x12c1, seq=1/256, ttl=1 (no response found!)
72	40.768386835	192.168.2.151	193.136.9.240	ICMP	1263	Echo (ping) request id=0x12c1, seq=2/512, ttl=1 (no response found!)

Figura 2.8: Tráfego capturado pelo *Wireshark* para os comandos (i) e (ii).

Os restantes são referentes ao tráfego do comando (ii), com pacotes de tamanho superior que se encontram fragmentados, assunto falado de seguida.

Nota: A diferença de tamanho referida pode ser vista do registo 69 para a frente.

Partindo da primeira mensagem ICMP capturada (para o comando (i), ou seja, com o tamanho estabelecido por defeito), obteve-se o seguinte pacote:

```

No.      Time      Source      Destination      Protocol Length Info
 6 6.072424202 192.168.2.151 193.136.9.240    ICMP      74      Echo (ping) request
id=0x12bf, seq=1/256, ttl=1 (no response found!)
Frame 6: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: AsustekC_0f:38:da (54:a0:50:0f:38:da), Dst: Vmware_5e:60:ad (00:0c:29:5e:60:ad)
Internet Protocol Version 4, Src: 192.168.2.151, Dst: 193.136.9.240
0100 .... = Version: 4
... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 60
Identification: 0xd971 (55665)
Flags: 0x0000
Time to live: 1
Protocol: ICMP (1)
Header checksum: 0x5198 [validation disabled]
[Header checksum status: Unverified]
Source: 192.168.2.151
Destination: 193.136.9.240
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0x6fba [correct]
[Checksum Status: Good]
Identifier (BE): 4799 (0x12bf)
Identifier (LE): 48914 (0xbf12)
Sequence number (BE): 1 (0x0001)
Sequence number (LE): 256 (0x0100)
[No response seen]
Data (32 bytes)
0000 48 49 4a 4b 4c 4d 4e 4f 50 51 52 53 54 55 56 57  HIJKLMNPQRSTUUVW
0010 58 59 5a 5b 5c 5d 5e 5f 60 61 62 63 64 65 66 67  XYZ[\]^_`abcdefg

```

Figura 2.9: Primeiro pacote ICMP para o comando (i).

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

R: O endereço IP da interface ativa do nosso computador é o *Source* **192.168.2.151** que representa de onde o pacote foi enviado.

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

R: O campo protocolo indica que tipo de dados podem ser encontrados na porção *payload* do datagrama IP. Neste caso o protocolo usado foi o ICMP (Internet Control Message Protocol).

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

R: O cabeçalho IP (header) tem 20 bytes como indicado no datagrama da figura acima.

Os dados (payload) representam 32 bytes do frame (Data (32 bytes) na expansão da parte do ICMP da imagem).

O cálculo do payload é relativamente simples: O frame capturado pelo Wireshark apresenta um tamanho de 74 bytes que corresponde à soma de 60 bytes de pacote (definido pelo traceroute por defeito) e 14 bytes de especificação ethernet (MAC Header).

Logo, 74 bytes = 14 bytes (ethernet spec.) + 60 bytes (pacote IP). O payload pode ser obtido a partir do pacote IP, sendo que, neste pacote de 60 bytes correspondem a: 20 bytes para o header IP, 8 bytes para o header ICMP e o resto serão os dados (payload).

Assim, o payload é de $60 - 20 - 8 = 32$ bytes.

d) *"O datagrama IP foi fragmentado? Justifique."*

R: Não foi fragmentado. Este facto pode ser provado por diversas razões, entre elas:

- A quantidade de dados enviados por cada pacote é menor que o MTU (Maximum Transmission Unit = 1500 bytes (payload)), sendo apenas 60 bytes;
- Todas as flags do datagrama original estão a zero ("Flags: 0x0000") o que indica que o campo "More fragments" também está a zero o que indica que este frame é o último (neste caso o único).

e) *"Ordene os pacotes capturados de acordo com o endereço IP fonte (e.g., seleccionando o cabeçalho da coluna Source), e analise a sequência de tráfego ICMP gerado a partir do endereço IP atribuído à interface da sua máquina. Para a sequência de mensagens ICMP enviadas pelo seu computador, indique que campos do cabeçalho IP variam de pacote para pacote."*

R: Para ordenar os pacotes de acordo com o endereço fonte pressionamos o cabeçalho da coluna *Source*. Analisando apenas as mensagens enviadas pela interface do nosso computador, ou seja, com o IP 192.168.2.151, observamos as seguintes variações no header IP:

- O campo **TTL** varia, consequência de aumentar em 1 o seu valor em cada mensagem enviada pelo traceroute;
- O campo **Identification** também varia, pois este serve para indicar a sua identificação para outros fragmentos deste pacote na rede, o que não se verifica;
- Por fim, o campo **Header Checksum** também varia de mensagem para mensagem e é usada pelo router para verificar a validade do pacote.

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

R: O valor do campo identificação vai aumentando em um (em hexadecimal) a cada mensagem nova. O valor do campo TTL aumenta de um em um a cada 3 mensagens de *request*.

g) "Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP TTL exceeded enviadas ao seu computador. Qual é o valor do campo TTL? Esse valor permanece constante para todas as mensagens de resposta ICMP TTL exceeded enviados ao seu host? Porquê?"

R: Para ordenar os pacotes de acordo com o endereço destino pressionamos o cabeçalho da coluna Destination.

Analisando apenas as mensagens enviadas à interface do nosso computador, verificamos que o valor do campo TTL não é constante: Tem valor de 64 nas primeiras 3 mensagens de erro do *Source 192.168.2.1* que se encontra a **um hop** do nosso host e passa para TTL = 254 nas mensagens do *Source 193.136.19.254* que se encontra a **dois hops**.

Na teoria, este valor está associado a padrões *default* definidos por diferentes Sistemas Operativos como pode ser visto neste *website*: "<https://subinsb.com/default-device-ttl-values/>" o que nos leva a pensar que ao passar por diferentes *gateways* esse valor tenha vindo a ser alterado por diferentes sistemas. Na tabela seguinte apresentam-se alguns exemplos:

Valores de TTL padrão		
Device / OS		TTL
*nix	(Linux/U-nix)	64
Windows		128
Solaris/AIX		254

Pergunta 3

"Pretende-se agora analisar a fragmentação de pacotes IP. (...) Observe o tráfego depois do tamanho de pacote ter sido definido para 42XX bytes."

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

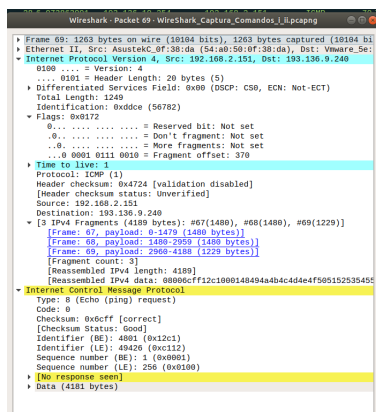


Figura 2.10: Primeiro pacote ICMP capturado para o comando (ii).

R: A primeira mensagem ICMP para um tamanho de pacote definido para 4209 bytes, apresentada acima na figura 2.10 com o registo 69 do *Wireshark* indica que o pacote foi fragmentado pois as *flags*, nomeadamente, a "fragment offset", assume um valor diferente de zero, correspondendo ao deslocamento deste frame em relação aos outros fragmentos. A fragmentação foi necessária pois a quantidade de dados enviados é maior que o MTU (Maximum Transmission Unit = 1500 bytes (payload)).

b) "Imprima o primeiro fragmento do datagrama IP segmentado. Que informação no cabeçalho indica que o datagrama foi fragmentado? Que informação no cabeçalho IP indica que se trata do primeiro fragmento? Qual é o tamanho deste datagrama IP?"

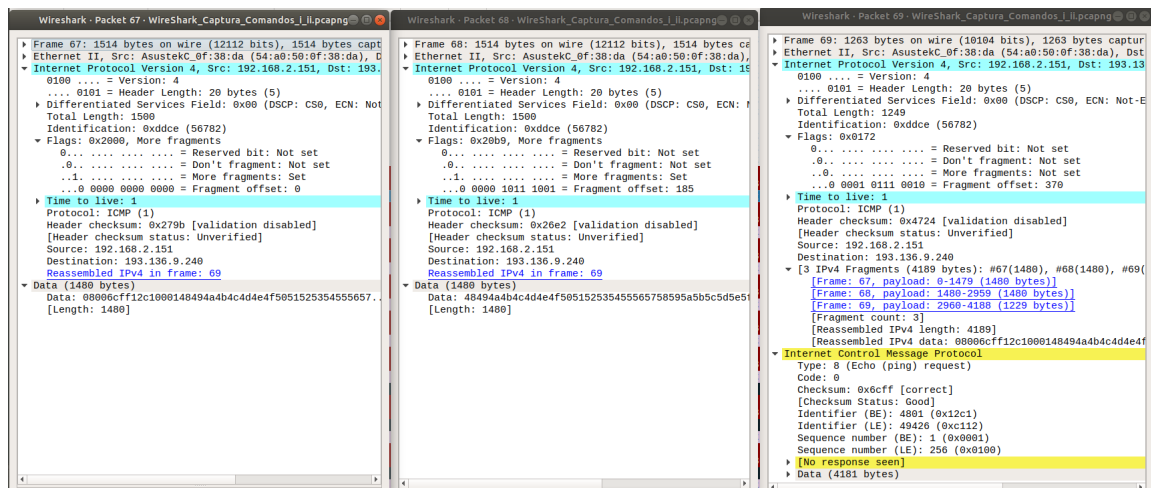


Figura 2.11: Fragmentos do primeiro pacote de 4209 bytes capturado.

R: O primeiro fragmento corresponde à primeira janela da figura 2.11 cujo registo é o 67. A informação que indica que foi fragmentado foi a seguinte:

- A flag *more fragments* encontra-se a 1 o que nos permite afirmar que existem mais segmentos para além deste;

Trata-se do primeiro fragmento porque:

- A flag *Fragment offset* tem o valor 0, ou seja este é o primeiro fragmento.

E, por fim, este datagrama IP tem 1500 bytes de *payload* do frame *ethernet*, nos quais, 1480 correspondem a dados (lixo), e 20 bytes ao *header* do IP (na pergunta e) analisaremos este valor com mais detalhe).

c) "Imprima o segundo fragmento do datagrama IP original. Que informação do cabeçalho IP indica que não se trata do 1o fragmento? Há mais fragmentos? O que nos permite afirmar isso?"

R: O fragmento pedido encontra-se na figura 2.11 como sendo a segunda janela correspondendo ao registo 68 do *Wireshark*. Não se trata do primeiro fragmento porque a flag "Fragment offset" é diferente de zero, neste caso toma o valor de 185. Existem mais fragmentos porque a flag "More fragments" é igual a 1, o que indica que este (68) não é o último fragmento.

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

R: Foram criados 3 fragmentos a partir do datagrama original. O registo 69 da figura 2.11 corresponde ao último fragmento pois a *flag* "More fragments" toma o valor 0.

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

R: Os campos que mudam no cabeçalho IP entre os diferentes fragmentos são os seguintes:

- O campo ***Total Length*** que corresponde ao tamanho do datagrama IP sendo de 1500 bytes no fragmento 67 e 68, e de 1249 no fragmento 69. Esta informação é um pouco contraditória pois a soma dos 3, segundo o *Wireshark* é de 4249 ao que se deve retirar 20 bytes do *header* do IP do 67 e 20 bytes do 68, pelo que, o *header* final vai apenas no registo 69 onde está indicado o tamanho de cada fragmento (1480 bytes para o 67, 1480 bytes para o 68 e 1239 bytes para o 69);
- A **flag** More fragments que no registo 67 e 68 indicam que estes fragmentos não são os últimos e no 69 já se encontra a 0;
- A **flag** Fragment Offset que no registo 67 é de 0, no registo 68 é de 185 e no registo 69 é de 370. Por exemplo, o valor 185 indica que o fragmento 2 começa (185 * 8) bytes à frente do fragmento 1.

Fim da Parte I

Capítulo 3

Parte II

3.1 Questões e Respostas

Enunciado

”Considere que a organização MIEI-RC é constituída por quatro departamentos (A, B, C e D) e cada departamento possui um router de acesso à sua rede local. Estes routers de acesso (Ra, Rb, Rc, e Rd) estão interligados entre si por ligações Ethernet a 1 Gbps, formando um anel. Por sua vez, existe um servidor (S1) na rede do departamento A e (...) Construa uma topologia CORE que reflita a rede local da empresa. Para facilitar a visualização pode ocultar o endereçamento IPv6.”

Caso de estudo:

A topologia de rede definida, com base nas indicações do enunciado, foi a seguinte:

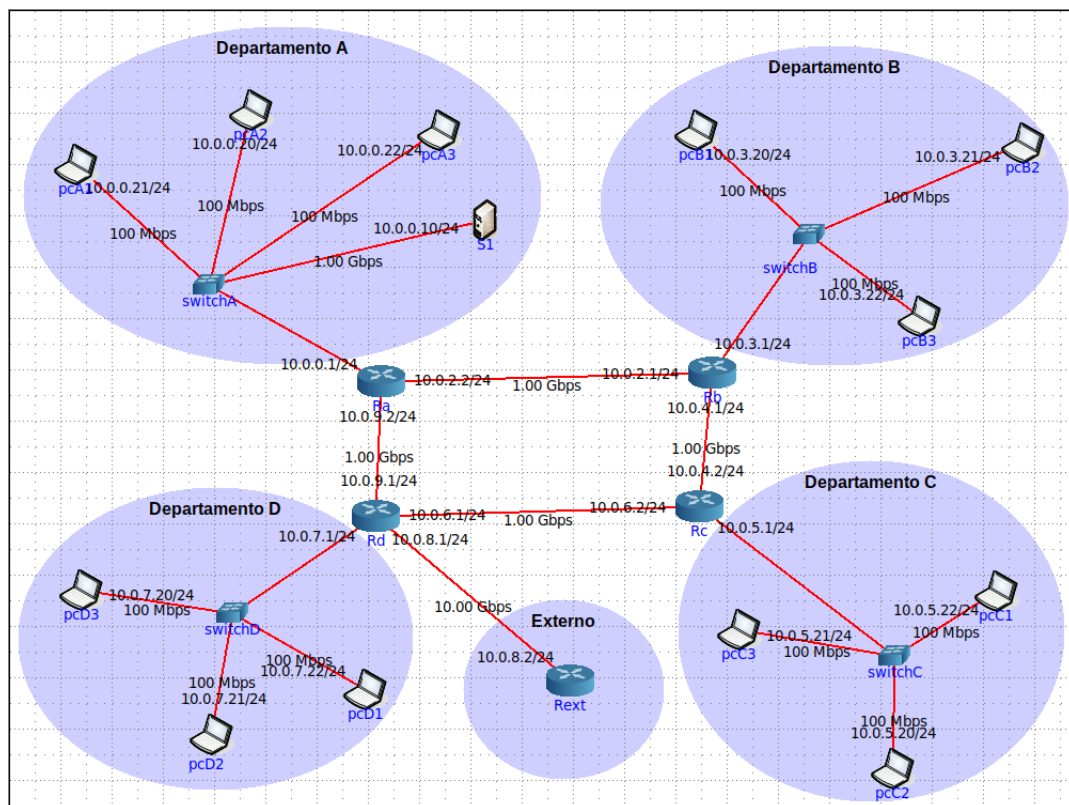


Figura 3.1: Topologia de rede MIEI-RC.

Pergunta 1

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

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

R: Para facilitar a resposta, a figura 3.1 indica, de forma clara, a topologia definida, os diferentes IPs atribuídos e as diferentes redes formadas.

b) *"Trata-se de endereços públicos ou privados? Porquê?"*

R: São endereços privados. Podemos dizer que existem endereços públicos designados para uso em redes que nos permitem aceder à Internet, no entanto, existem blocos de endereços que pouco ou nada necessitam de acesso à Internet. Os endereços gerados pelo CORE são privados e pertencem ao bloco cujo intervalo de endereços varia entre **10.0.0.0** e **10.255.255.255** (10.0.0.0/8), que seguem o padrão RFC 1918.

c) *"Por que razão não é atribuído um endereço IP aos switches?"*

R: Na realidade, um switch como o usado na topologia definida no exercício não necessita de saber o que é um endereço IP e pertence a outro *layer* do modelo OSI, o *layer 2: data link layer*. Precisa, ao invés, dos *MAC address* dos elementos que compõem a rede à qual o switch está ligado. Tem, assim, a simples função de receber pacotes e distribuí-los pelos diferentes componentes que a compõem.

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

R: Com base no enunciado desta questão, bastou correr o seguinte comando num dos laptops de cada departamento:

`$ping 10.0.0.10`

Tendo obtido resposta positiva por parte de todos os departamentos, como se pode comprovar com as figuras seguintes:

```
root@pcA1:/tmp/ycrcore.42071/pcA1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_seq=1 ttl=64 time=0.064 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=64 time=0.034 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=64 time=0.030 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 203ms
rtt min/avg/max/ndev = 0.034/0.064/0.064/0.014 ms
root@pcA1:/tmp/ycrcore.42071/pcA1.conf#
```

(a) pcA - S1

```
root@pcB1:/tmp/ycrcore.42071/pcB1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_seq=1 ttl=62 time=0.134 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=62 time=0.050 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=62 time=0.044 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 202ms
rtt min/avg/max/ndev = 0.044/0.077/0.134/0.041 ms
root@pcB1:/tmp/ycrcore.42071/pcB1.conf#
```

(b) pcB - S1

Figura 3.2: Execução do comando ping entre os laptops dos departamentos A e B e o servidor.

```
root@pcC1:/tmp/ycrcore.42071/pcC1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_seq=1 ttl=61 time=0.102 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=61 time=0.068 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=61 time=0.064 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 202ms
rtt min/avg/max/ndev = 0.064/0.078/0.102/0.017 ms
root@pcC1:/tmp/ycrcore.42071/pcC1.conf#
```

(a) pcC - S1

```
root@pcD1:/tmp/ycrcore.42071/pcD1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_seq=1 ttl=62 time=0.086 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=62 time=0.055 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=62 time=0.056 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 202ms
rtt min/avg/max/ndev = 0.055/0.065/0.086/0.017 ms
root@pcD1:/tmp/ycrcore.42071/pcD1.conf#
```

(b) pcD - S1

Figura 3.3: Execução do comando ping entre os laptops dos departamentos C e D e o servidor.

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

R: Existe conectividade do Rext e o servidor e tal facto pode ser comprovado pela seguinte figura:

```
root@Rext:/tmp/pycore.42071/Rext.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data:
64 bytes from 10.0.0.10: icmp_seq=1 ttl=62 time=0.055 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=62 time=0.057 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=62 time=0.063 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2045ms
rtt min/avg/max/mdev = 0.055/0.058/0.063/0.007 ms
```

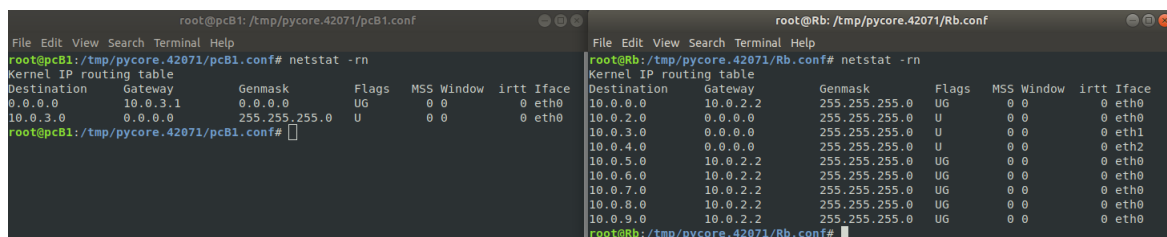
Figura 3.4: Execução do comando ping no Rext para o servidor 1.

Pergunta 2

”Para o router e um laptop do departamento B:”

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

R: As tabelas de encaminhamento calculadas com o comando `netstat -rn` são as seguintes:



```
root@pcB1:/tmp/pycore.42071/pcB1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.3.1 0.0.0.0 UG 0 0 0 eth0
10.0.3.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0

root@Rb:/tmp/pycore.42071/Rb.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth0
10.0.2.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.3.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
10.0.5.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth0
10.0.6.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth0
10.0.7.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth0
10.0.8.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth0
10.0.9.0 10.0.2.2 255.255.255.0 UG 0 0 0 eth0
```

Figura 3.5: Esquerda: `netstat` no pcB1; Direita: `netstat` no router Rb.

A tabela de encaminhamento indica-nos na coluna **Destination** as redes destino possíveis para as quais existe um **Gateway** respetivo que corresponde ao próximo destino que um pacote deve tomar para aceder a essa rede.

Análise da tabela do laptop B1 (pcB1): Nesta tabela temos duas linhas: A primeira linha indica um destino 0.0.0.0 que corresponde a um valor **default**, isto é, quando o pcB1 recebe um pacote e não sabe qual o destino final do mesmo, envia-o para o Gateway 10.0.1.1 onde, este último, tratará de reencaminhar o pacote para o seu eventual destino; Na segunda linha temos um destino (não default) 10.0.1.0, que corresponde à rede do departamento A dada pelos primeiros 3 bytes do endereço IP (10.0.1) cuja máscara é 255.255.255.0 (/24), ou seja, um pacote que o pcB1 receba com destino à rede do departamento A então este já se encontra na rede porque o respetivo valor do Gateway é "0.0.0.0".

Análise da tabela do router B (Rb): A análise da tabela de encaminhamento é, de certo modo, parecida com a análise anterior, exceto no facto de que um router, normalmente, não possui um valor **default** pois sabe sempre o destino associado a um dado pacote. Observando uma das linhas, por exemplo, a linha 1 vemos que para um pacote ir para o destino 10.0.0.0 terá de utilizar o Gateway 10.0.2.2 cuja máscara é 255.255.255.0 e sairá pela interface eth0 do router. Por outro lado, vejamos a linha 2, para um pacote ir para o destino 10.0.2.0 não precisa de sair da rede pois já se encontra na mesma, isto é, o respetivo Gateway é "0.0.0.0".

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

R: O encaminhamento que está a ser utilizado é dinâmico. Partindo da sugestão do enunciado e analisando os processos que estão a correr num router, por exemplo, em Ra:

Figura 3.6: Execução do comando *top* para análise de processos a correr em Ra.

Conseguimos verificar a existência de processos cujo *USER* é o **quagga** e está a executar programas que providenciam a implementação de protocolos de encaminhamentos associados a encaminhamento dinâmico (como o OSPF - *Open Shortest Path First*).

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

R: Esta alínea pedia para efetuar a remoção da rota por defeito definida no servidor S1 do departamento A. Para tal segue-se uma figura com a execução do comando indicado:

Figura 3.7: Tabela de encaminhamento antes e depois da execução do comando *route delete*.

Quando retirada a rota por defeito da tabela de encaminhamento do servidor 1 percebemos que o servidor 1 fica inacessível por parte de todos os utilizadores da empresa que não se encontrem no departameto A, visto que quando tentamos enviar pacotes de teste através do comando **ping** dos vários utilizadores para o servidor 1 não obtemos resposta.

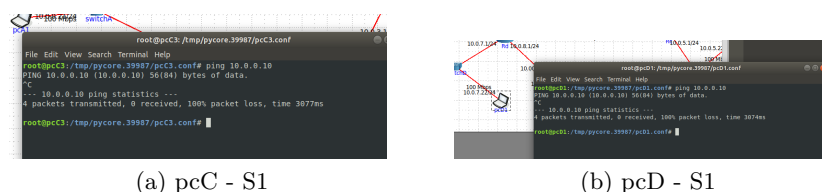


Figura 3.8: Pacotes perdidos no envio de teste entre laptops do departamento C e D com o Servidor 1.

O que deduzimos que estará a acontecer é o facto de que ao enviar um pacote de um laptop, por exemplo, do departamento B para o servidor, o mesmo chega ao Servidor mas não sabe como voltar pois a rota por defeito foi removida e, assim, o laptop em B não recebe a resposta mas sim deduz que o pacote foi perdido no envio, pelos outputs dados pela execução do comando.

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

R: Para restaurar a conectividade com o servidor S1 é necessário adicionar linhas à sua tabela de encaminhamento que indiquem como um pacote que chegue a S1 possa regressar a uma qualquer rede deste sistema, ou seja, teremos de adicionar os destinos possíveis associados a um Gateway único, Ra, que depois envie o pacote para o seu destino final. Foram então adicionados os seguintes comandos, com o seguinte padrão:

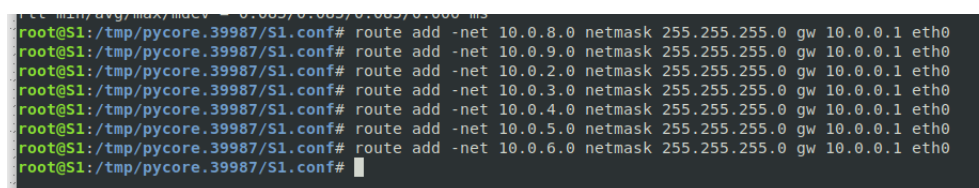
Regra geral:

```
$route add -net <rede_destino> netmask 255.255.255.0 gw 10.0.0.1 eth0
```

Exemplo:

```
$route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
```

Ou seja, adicionar uma entrada à tabela com a rede destino 10.0.7.0 cuja **netmask** é sempre a mesma para o Gateway do Router A (10.0.0.1) pela interface **eth0**. Seguem-se os outros comandos adicionados:

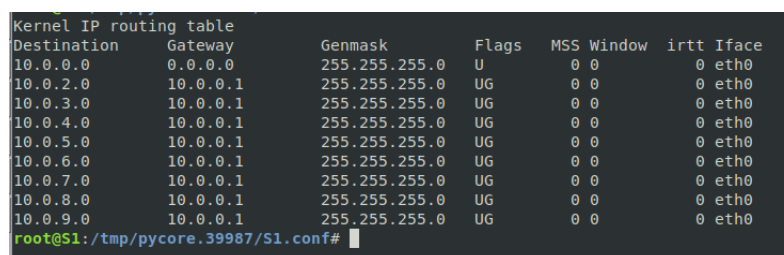


```
root@S1:/tmp/pycore.39987/S1.conf# route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
root@S1:/tmp/pycore.39987/S1.conf# route add -net 10.0.9.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
root@S1:/tmp/pycore.39987/S1.conf# route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
root@S1:/tmp/pycore.39987/S1.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
root@S1:/tmp/pycore.39987/S1.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
root@S1:/tmp/pycore.39987/S1.conf# route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
root@S1:/tmp/pycore.39987/S1.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.0.1 eth0
root@S1:/tmp/pycore.39987/S1.conf#
```

Figura 3.9: Comandos inseridos para restaurar a conectividade com S1.

d) *"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."*

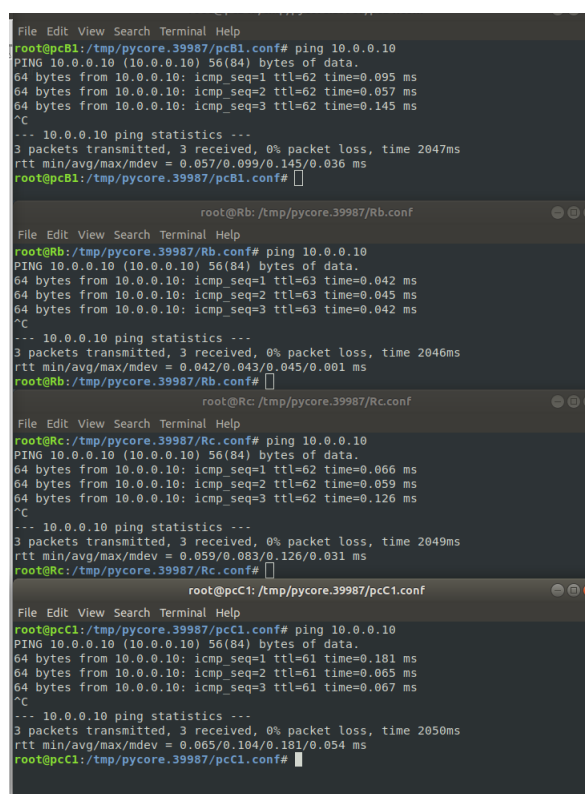
R: Com a inserção dos comandos descritos anteriormente otivemos um nova tabela de encaminhamento do servidor 1:



```
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
10.0.2.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.3.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.4.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.5.0 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 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.8.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
10.0.9.0 10.0.0.1 255.255.255.0 UG 0 0 0 eth0
root@S1:/tmp/pycore.39987/S1.conf#
```

Figura 3.10: Tabela de encaminhamento de S1.

Com esta tabela a conectividade foi restaurada como se pode observar por alguns exemplos de testes na figura seguinte:



```
File Edit View Search Terminal Help
root@pcB1:/tmp/pycore.39987/pcB1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=62 time=0.095 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=62 time=0.057 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=62 time=0.145 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2047ms
rtt min/avg/max/mdev = 0.057/0.099/0.145/0.036 ms
root@pcB1:/tmp/pycore.39987/pcB1.conf#

File Edit View Search Terminal Help
root@Rb:/tmp/pycore.39987/Rb.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=63 time=0.042 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=63 time=0.045 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=63 time=0.042 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2046ms
rtt min/avg/max/mdev = 0.042/0.043/0.045/0.001 ms
root@Rb:/tmp/pycore.39987/Rb.conf#

File Edit View Search Terminal Help
root@Rc:/tmp/pycore.39987/Rc.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=62 time=0.066 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=62 time=0.059 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=62 time=0.126 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2049ms
rtt min/avg/max/mdev = 0.059/0.083/0.126/0.031 ms
root@Rc:/tmp/pycore.39987/Rc.conf#

File Edit View Search Terminal Help
root@pcC1:/tmp/pycore.39987/pcC1.conf# ping 10.0.0.10
PING 10.0.0.10 (10.0.0.10) 56(84) bytes of data.
64 bytes from 10.0.0.10: icmp_seq=1 ttl=61 time=0.181 ms
64 bytes from 10.0.0.10: icmp_seq=2 ttl=61 time=0.065 ms
64 bytes from 10.0.0.10: icmp_seq=3 ttl=61 time=0.067 ms
^C
--- 10.0.0.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2050ms
rtt min/avg/max/mdev = 0.065/0.104/0.181/0.054 ms
root@pcC1:/tmp/pycore.39987/pcC1.conf#
```

Figura 3.11: Alguns testes de conectividade com S1.

Pergunta 3

”Por forma a minimizar a falta de endereços IPv4 é comum a utilização de sub-redes.(...)Para definir endereços de sub-rede é necessário usar a parte prevista para endereçamento de host, não sendo possível alterar o endereço de rede original.(...)Assuma que o endereçamento entre os routers se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.”

a) *”Considere que dispõe apenas do endereço de rede IP 172.yyx.32.0/20, em que “yy” são os dígitos correspondendo ao seu número de grupo (Gyy) e “x” é o dígito correspondente ao seu turno prático (PLx). 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.”*

R: Partindo do endereço de rede IP 172.yyx.32.0 onde yy corresponde a 9 (o número do nosso grupo) e x corresponde ao número 6 (turno PL6), começamos a desenvolver o nosso novo esquema de rede:

O primeiro passo foi estender a máscara em alguns bits para suportar todas as redes dos departamentos. Percebemos então que bastaria estender a máscara em dois bits que corresponderiam a $2^2 = 4$ subredes. Esta divisão permite a criação de $2^{10} - 2 = 1022$ hosts por cada subrede.

Na tabela a seguir apresentada podemos verificar os bits escolhidos para **subnetting** presentes na terceira coluna da representação do endereço, a coluna $32 = 0010|00|00$ onde os bits “|00|” serão usados para *subnetting*.

bits para <i>subnetting</i>				
Decimal	172.	96.	32.	0
172.96.32.0	10101100.	01100000.	0010 00 00.	00000000

Esta divisão permite criar as diferentes subredes referidas de seguida:

Subredes				
172.96.32.0	10101100.	01100000.	0010 00 00.	00000000
172.96.36.0	10101100.	01100000.	0010 01 00.	00000000
172.96.40.0	10101100.	01100000.	0010 10 00.	00000000
172.96.44.0	10101100.	01100000.	0010 11 00.	00000000

A máscara utilizada passa a ser de 22 bits (/22 em CIDR). Assim, após definir as subredes só falta atribuir os IPs aos diferentes componentes das redes dos departamentos. Para isso definimos o seguinte padrão (neste exemplo usar-se-á a rede 172.96.32.0):

- 172.96.32.1 até 172.96.32.9 *routers*;
- 172.96.32.10 até 172.96.32.19 *servers*;
- 172.96.32.20 até 172.96.35.254 *hosts*;

A seguinte imagem mostra, na prática, a atribuição dos IPs para as diferentes subredes:

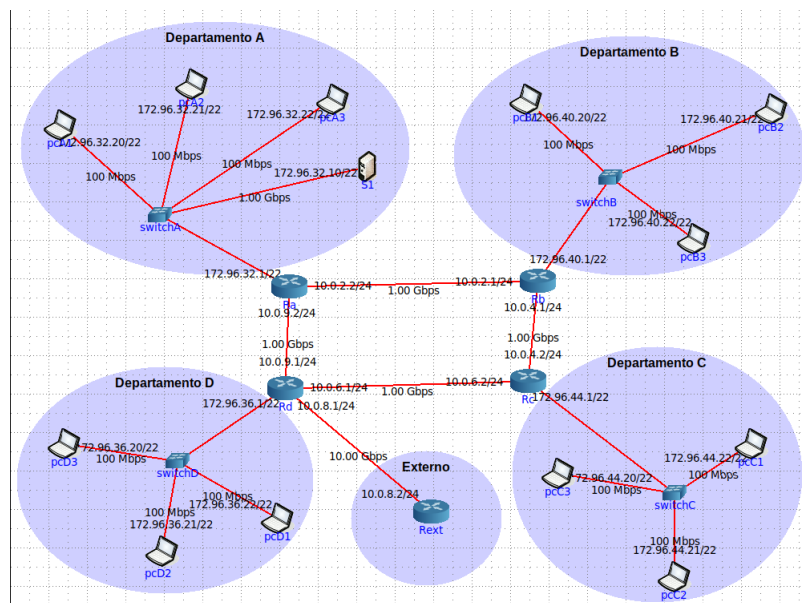


Figura 3.12: Topologia de rede dos departamentos com o novo endereçamento.

b) "Qual a máscara de rede que usou (em notação decimal)? Quantos interfaces IP pode interligar em cada departamento? Justifique."

R: Como referido na questão anterior, a máscara de rede utilizada foi a 255.255.252.0 cuja notação em CIDR corresponde a /22. Esta máscara de rede utiliza mais 2 bits do que a máscara inicial de 20 bits para representar as diferentes subredes. É possível utilizar mais bits para *subnetting* desde que garantamos um mínimo de 3 bits para suportar o número de hosts por subrede que, neste caso, no máximo pode ser de 5 interfaces (no caso do departamento A).

Com esta notação utilizamos 10 bits (32 bits total - 22 bits rede = 10 bits host) para representar os diferentes hosts da rede e teremos de excluir aqueles que representam os endereços reservados *bits todos a um* ou *bits todos a zero*. Assim sendo, temos $2^{10} - 2 = 1022$ interfaces IP em cada departamento o que dá um total de $1022 * 4 = 4088$ interfaces IP.

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

R: Nesta última questão bastou testar o comando *ping* entre os diferentes hosts dos 4 departamentos, como se pode ver de seguida:

```

root@pcA1:/tmp/pycore-42699/pcA1.conf
root@pcA1:/tmp/pycore-42699/pcA1.conf# ping 172.96.36.20
PING 172.96.36.20 (172.96.36.20) 56(84) bytes of data:
64 bytes from 172.96.36.20: icmp_seq=1 ttl=62 time=0.117 ms
64 bytes from 172.96.36.20: icmp_seq=2 ttl=62 time=0.063 ms
64 bytes from 172.96.36.20: icmp_seq=3 ttl=62 time=0.064 ms
^C
... 172.96.36.20 ping statistics ...
3 packets transmitted, 3 received, 0% packet loss, time 203ms
rtt min/avg/max/mdev = 0.063/0.081/0.117/0.026 ms
root@pcA1:/tmp/pycore-42699/pcA1.conf#

```

(a) pcA1 - pcD3

```

root@pcB2:/tmp/pycore-42699/pcB2.conf
root@pcB2:/tmp/pycore-42699/pcB2.conf# ping 172.96.44.21
PING 172.96.44.21 (172.96.44.21) 56(84) bytes of data:
64 bytes from 172.96.44.21: icmp_seq=1 ttl=62 time=0.115 ms
64 bytes from 172.96.44.21: icmp_seq=2 ttl=62 time=0.062 ms
64 bytes from 172.96.44.21: icmp_seq=3 ttl=62 time=0.058 ms
^C
... 172.96.44.21 ping statistics ...
3 packets transmitted, 3 received, 0% packet loss, time 203ms
rtt min/avg/max/mdev = 0.059/0.079/0.115/0.026 ms
root@pcB2:/tmp/pycore-42699/pcB2.conf#

```

(b) pcB2 - pcC2

Figura 3.13: Execução do comando ping entre diferentes departamentos.

```

root@pcC2:/tmp/pycore-42699/pcC2.conf
root@pcC2:/tmp/pycore-42699/pcC2.conf# ping 172.96.32.22
PING 172.96.32.22 (172.96.32.22) 56(84) bytes of data:
64 bytes from 172.96.32.22: icmp_seq=1 ttl=61 time=0.185 ms
64 bytes from 172.96.32.22: icmp_seq=2 ttl=61 time=0.072 ms
64 bytes from 172.96.32.22: icmp_seq=3 ttl=61 time=0.075 ms
^C
... 172.96.32.22 ping statistics ...
3 packets transmitted, 3 received, 0% packet loss, time 204ms
rtt min/avg/max/mdev = 0.072/0.084/0.185/0.014 ms
root@pcC2:/tmp/pycore-42699/pcC2.conf#

```

(a) pcC2 - pcA3

```

root@pcD2:/tmp/pycore-42699/pcD2.conf
root@pcD2:/tmp/pycore-42699/pcD2.conf# ping 172.96.40.22
PING 172.96.40.22 (172.96.40.22) 56(84) bytes of data:
64 bytes from 172.96.40.22: icmp_seq=1 ttl=61 time=0.122 ms
64 bytes from 172.96.40.22: icmp_seq=2 ttl=61 time=0.072 ms
64 bytes from 172.96.40.22: icmp_seq=3 ttl=61 time=0.070 ms
^C
... 172.96.40.22 ping statistics ...
3 packets transmitted, 3 received, 0% packet loss, time 204ms
rtt min/avg/max/mdev = 0.070/0.088/0.122/0.025 ms
root@pcD2:/tmp/pycore-42699/pcD2.conf#

```

(b) pcD2 - pcB3

Figura 3.14: Execução do comando ping entre diferentes departamentos.

Mais se afirma que foi necessária atenção à forma como o CORE atribui os *gateway* dos diferentes hosts da rede, sendo que, o primeiro host de cada subrede é atribuído a esses *gateways*. Caso contrário seria necessário alterar as rotas *default* de cada subrede. Desta forma seguimos o padrão anteriormente referido.

Fim da Parte II

Capítulo 4

Conclusões

A elaboração deste trabalho permitiu aprimorar a vertente prática associada a esta Unidade Curricular no qual se inclui o estudo, numa primeira fase, do *Internet Protocol* e em como este protocolo se inclui no modelo OSI, isto é, na camada de rede deste modelo, e a sua função no encaminhamento de dados (datagramas IP) a partir dos *routers*.

Foi importante também a análise dos campos que compõem os diferentes datagramas IP que permitiram descobrir através da análise, usando programas apropriados como o *Wireshark*, a razão de existência do campo TTL num pacote e como programas como o *traceroute* utilizavam esses valores em seu proveito para criar a rota pela qual um pacote atravessaria até chegar a um dado destino. Para além do campo referido anteriormente, a fragmentação, também incluída na primeira fase, foi um assunto que, na prática, permitiu perceber a teoria estudada acerca dos limites associados aos componentes que compõem a rede e de que como se poderia proceder para rearranjar o pacote fragmentado a partir da informação presente nos diferentes fragmentos.

Na segunda parte deste trabalho utilizamos o conhecimento adquirido da primeira fase para escalar o endereçamento para uma rede propriamente dita. A necessidade de existirem endereços públicos e privados que são usados em contextos diferentes mas no fundo não deixam de servir para simples endereçamento. Os diversos testes de conectividade realizados a partir dos sistemas criados nesta fase serviram de confirmação à teoria associada a este conceito.

Depois passamos para a análise de tabelas de endereçamento que nos permitiram estudar a forma como *routers* e *hosts* redirecionavam os pacotes que recebiam e a necessidade da existência de rotas *default* para situações em que não é conhecida, previamente, a rota a seguir pelos pacotes. Situações estas que nos levaram a gerar outras rotas para as diferentes redes para contornar a situação de não ter rotas por defeito.

O último tema analisado foi um dos mais importantes em todo este trabalho, o *subnetting*, ou seja, a criação de subredes para minimizar a falta de endereços IPv4 através da utilização de bits associados ao host. Esta operação permite um maior controlo sob a rede e do seu crescimento reduzindo a congestão da mesma.

Concluimos assim que foi bastante positiva a realização deste trabalho principalmente porque todos os conceitos adquiridos em livros e exposições teóricos foram postos em prática de forma objetiva e desafiante o que tornou este projeto mais interessante.