TP2: Protocolo IPv4

Sérgio Jorge, João Freitas, and Alexandre Martins

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a77730,a74814,a77523}@alunos.uminho.pt

1 Introdução

O principal objetivo deste trabalho é o aprofundamento de conhecimentos em protocolo IPv4 através do estudo do formato de um pacote/datagrama IP, fragmentação de pacotes IP e endereçamento e encaminhamento IP, utilizando ferramentas como *Wireshark*, *PingPlotter*, *TraceRoute*, *NetStat* e *CORE*.

2 Grupo I - Parte 1

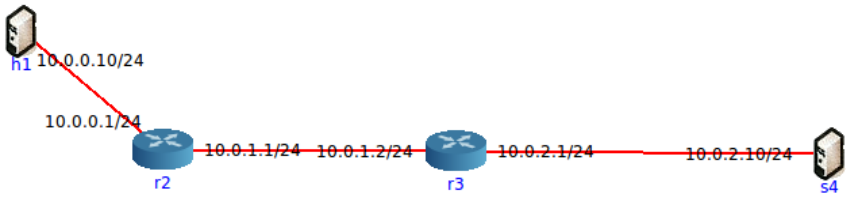

Figura 1: Topologia Inicial

2.1 Questão A

"Active o wireshark ou o tcpdump no pc h1. Numa shell de h1, execute o comando traceroute -I para o endereço IP do host s4."

```
11:23:27.037947 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:23:37.039445 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:23:37.052817 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:23:47.029813 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:23:47.040167 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:23:57.040950 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:23:57.066084 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:24:07.041410 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:24:07.045403 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:24:17.004338 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:24:17.043319 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:24:26.961535 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:24:27.044038 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:24:36.964813 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:24:37.045079 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:24:46.932197 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:24:47.047430 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:24:56.953402 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:24:57.048198 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:25:06.925931 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:25:07.048394 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:25:16.973043 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:25:17.049170 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
```

Figura 2: *TCPDump* no pc h1

```
root@h1:/tmp/pycore.33921/h1.conf# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1  A0 (10.0.0.1)  0.036 ms  0.007 ms  0.007 ms
 2  10.0.1.2 (10.0.1.2)  0.016 ms  0.011 ms *
 3  10.0.2.10 (10.0.2.10)  0.017 ms  0.014 ms  0.014 ms
```

Figura 3: *Traceroute* no host s4

2.2 Questão B

"Registe e analise o tráfego ICMP enviado por h1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado."

Como se pode observar na figura 4, temos varias sequências de *ICMP echo request* com a resposta correspondente *ICMP echo reply*. Pode-se também observar que a resposta do host s4 falha na 6ª sequência, daqui assumimos que o Time-To-Live (TTL) era menor que o número de saltos necessários para a ligação. Como na sequência 7 já se obteve resposta do host s4 e o TTL vai sendo incrementado em uma unidade para se descobrir qual o mínimo necessário, então, aqui, conclui-se que já é suficiente para efetuar a ligação com sucesso entre h1 e s4. Daí em diante, todas as tentativas de ligação obtém resposta com sucesso.

```
8
11:33:02.912981 IP 10.0.0.10 > 10.0.2.10: ICMP echo request, id 88, seq 6, lengt
h 40
11:33:02.912992 IP 10.0.1.2 > 10.0.0.10: ICMP time exceeded in-transit, length 6
8
11:33:02.913023 IP 10.0.0.10 > 10.0.2.10: ICMP echo request, id 88, seq 7, lengt
h 40
11:33:02.913065 IP 10.0.2.10 > 10.0.0.10: ICMP echo reply, id 88, seq 7, length
40
11:33:02.913100 IP 10.0.0.10 > 10.0.2.10: ICMP echo request, id 88, seq 8, lengt
h 40
11:33:02.913118 IP 10.0.2.10 > 10.0.0.10: ICMP echo reply, id 88, seq 8, length
40
11:33:02.913151 IP 10.0.0.10 > 10.0.2.10: ICMP echo request, id 88, seq 9, lengt
h 40
11:33:02.913168 IP 10.0.2.10 > 10.0.0.10: ICMP echo reply, id 88, seq 9, length
40
11:33:07.071879 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:33:07.088639 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
11:33:07.928813 ARP, Request who-has 10.0.0.10 tell 10.0.0.1, length 28
11:33:07.928869 ARP, Reply 10.0.0.10 is-at 00:00:00:aa:00:00, length 28
11:33:17.080079 IP6 fe80::200:ff:feaa:1 > ff02::5: OSPFv3, Hello, length 36
11:33:17.088845 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
```

Figura 4: Tráfego ICMP - entre h1 e s4

2.3 Questão C

"Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino s4? Verifique na prática que a sua resposta está correta."

O valor inicial mínimo do campo TTL tem de ser 3, visto que o número de saltos é 3 (h1->r2; r2->r3; r3->s4).

Além disso, na parte 2 da figura 5, procurou-se limitar o número de saltos em 3, e o destino foi alcançado na mesma.

```
root@h1:/tmp/pycore.33921/h1.conf# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1  A0 (10.0.0.1)  0.036 ms  0.007 ms  0.007 ms
 2  10.0.1.2 (10.0.1.2)  0.016 ms  0.011 ms *
 3  10.0.2.10 (10.0.2.10)  0.017 ms  0.014 ms  0.014 ms
root@h1:/tmp/pycore.33921/h1.conf#
root@h1:/tmp/pycore.33921/h1.conf# traceroute -I 10.0.2.10 -m 3
traceroute to 10.0.2.10 (10.0.2.10), 3 hops max, 60 byte packets
 1  A0 (10.0.0.1)  0.056 ms  0.001 ms  0.010 ms
 2  10.0.1.2 (10.0.1.2)  0.038 ms  0.016 ms  0.013 ms
 3  10.0.2.10 (10.0.2.10)  0.044 ms  0.020 ms  0.020 ms
```

Figura 5: *Traceroute* entre h1 e s4

2.4 Questão D

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

O valor médio do tempo ida-e-volta é 0.028ms, obtido através da média da terceira linha da Figura 5, somando os três tempos dados pelo terceiro salto e dividindo por 3.

3 Grupo II - Parte 1

3.1 Questão A

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

O endereço de IP da interface ativa é 192.168.2.190.

3.2 Questão B

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

O valor do campo protocolo é ICMP, *Internet Control Message Protocol*. Este protocolo é utilizado para reportar e gerir erros na rede. Ou seja, a partir dele, um *router* ou um host pode mencionar à origem que houve um erro no processamento do datagrama. Por exemplo:

- *time exceeded message* - quando o TTL chega a 0 e o datagrama tem de ser descartado;
- *echo request/reply* - mensagens enviadas para teste ou controlo da rede (*request*); se a máquina destino está disponível, então responde (*reply*);
- *destination unreachable* - quando não se consegue alcançar o destino;

É, também, um protocolo que funciona ao nível de rede e, por isso, as mensagens são encapsuladas em datagramas IP.

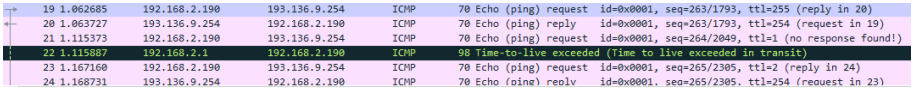

Figura 6: Primeiras mensagens ICMP

3.3 Questão 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?"

Verifica-se que o *header* do datagrama tem 20 bytes e no total, este tem 56 bytes. Por isso, fazendo 56-20 chega-se ao tamanho do *payload* que tem então 36 bytes.

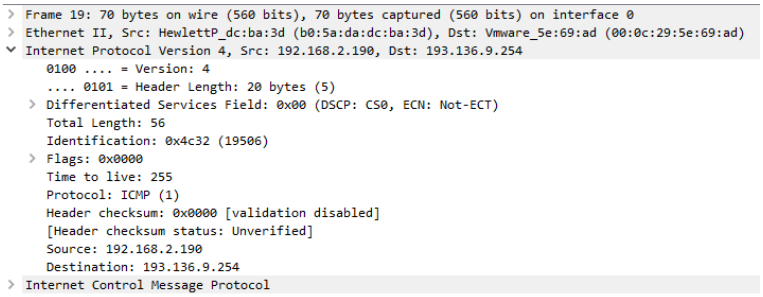

Figura 7: Datagrama IP

3.4 Questão D

"O datagrama IP foi fragmentado? Justifique."

A fragmentação surge quando um datagrama excede o MTU (Maximum Tranfer Unit). Como a flag *More Fragments* tem o valor *Not Set* e o *Fragment offset* é 0, podemos assumir que o datagrama IP não foi fragmentado. Ou seja, conclui-se que o MTU é superior ao tamanho do datagrama.

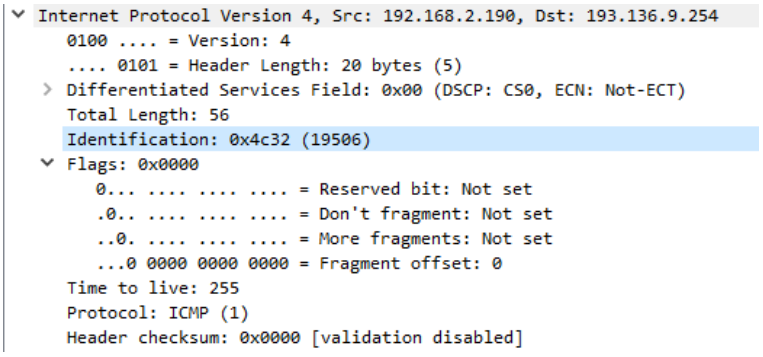

Figura 8

3.5 Questão E

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

Os campos do cabeçalho IP que variam são a identificação e o time-to-live (TTL).

A identificação muda porque identifica unicamente cada datagrama.

O TTL muda porque a máquina tenta ligar-se ao destino, começando com um TTL = 1. Quando o pacote é descartado, esta percebe que tem de incrementar o TTL até alcançar o destino.

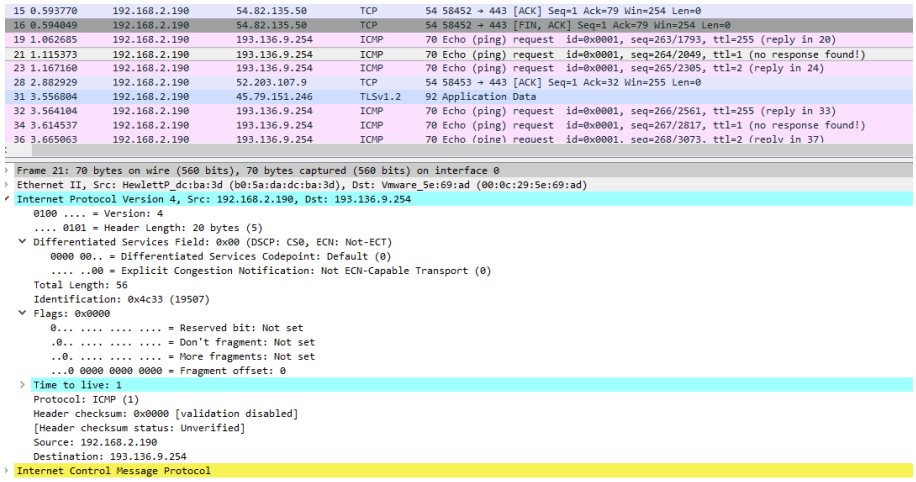

Figura 9: Tráfego ICMP

3.6 Questão F

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

O campo identificação muda, incrementando por um, e o campo Time-To-Live (TTL) começa em 255 porque este é o valor *default* em Linux. De seguida, vai testando quantos hops (saltos) são necessários para chegar ao destino, começando com TTL igual a 1 que dá a mensagem de erro *no response found!* e, de seguida, o TTL é incrementado para 2. Verifica-se, neste ponto, que já é possível alcançar o destino. Caso tal não acontecesse, o TTL continuaria a incrementar até saber o número de saltos mínimo para conseguir ligar-se ao destino.

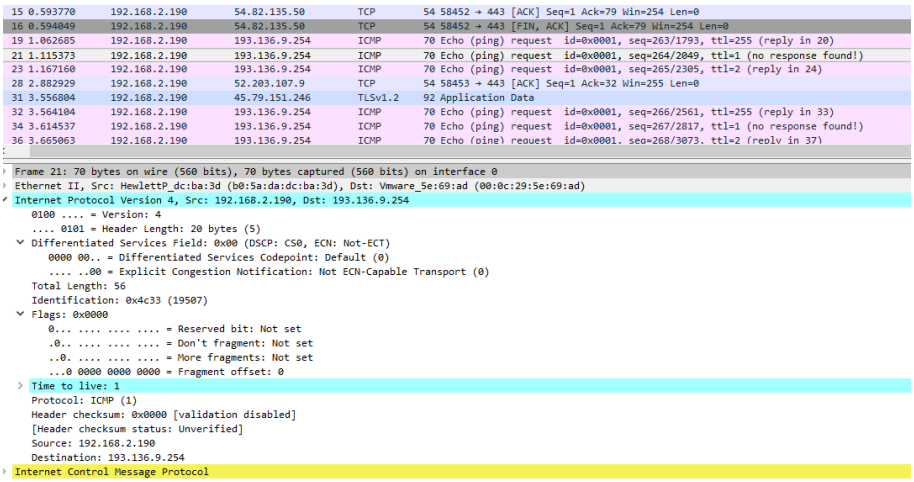

Figura 10: Tráfego ICMP

3.7 Questão G

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

O valor do campo Time-To-Live (TTL) é 64 visto que é o valor *default* da máquina destino e permanece constante entre todas as mensagens de resposta *ICMP TTL exceeded*. Por defeito, quando o *router* desconhece a distância ao host de destino, usa um TTL igual a 64, valor exageradamente alto, de forma a garantir a entrega do datagrama.

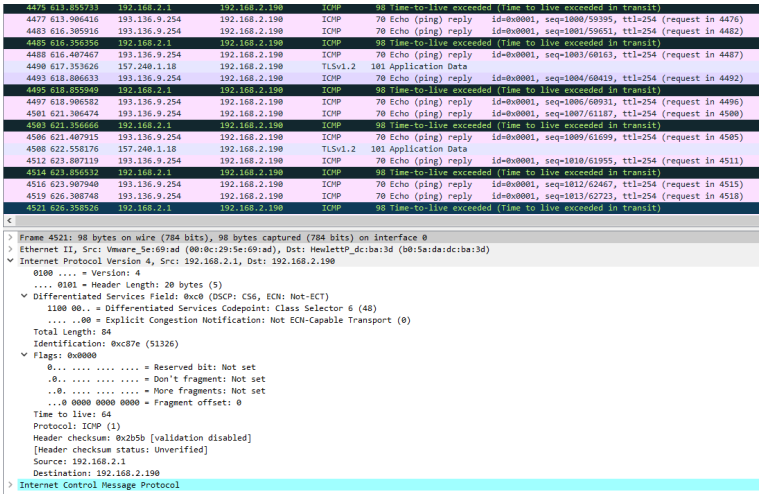

Figura 11: Tráfego ICMP

4 Grupo III - Parte 1

4.1 Questão A

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

Houve necessidade de fragmentar o pacote inicial porque este tem 3565 bytes de tamanho e excede o *Maximum Transmission Unit* (MTU) que é 1500 bytes.

4.2 Questão 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?"

Através da *flag More Fragments* que está assinalada a 1 e tem o valor *Set*, podemos afirmar que o datagrama foi fragmentado. Como o *Fragment offset* neste fragmento é 0 concluímos que se trata do primeiro fragmento. O tamanho deste datagrama é 1500 bytes, sendo que 20 destes bytes são para o *header* e 1480 para o *payload*.

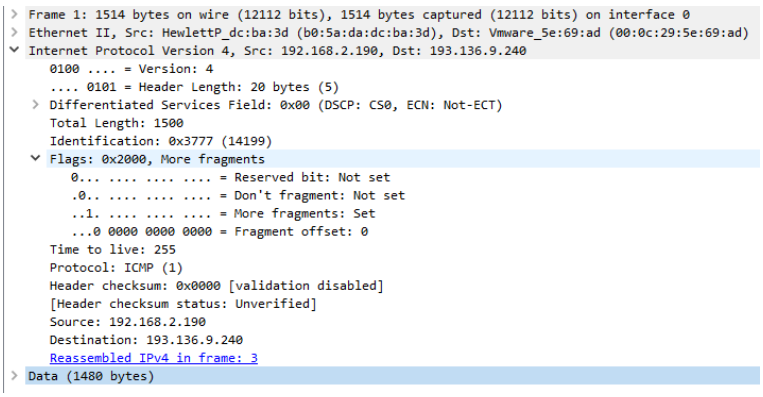

Figura 12: Fragmento 1

4.3 Questão 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?"

Contrariamente ao primeiro fragmento, o *Fragment offset* deste tem o valor 185, o que indica que não se trata do primeiro. Podemos afirmar que existem mais fragmentos porque a *flag More Fragments* está assinalada a 1 e tem o valor *Set*.

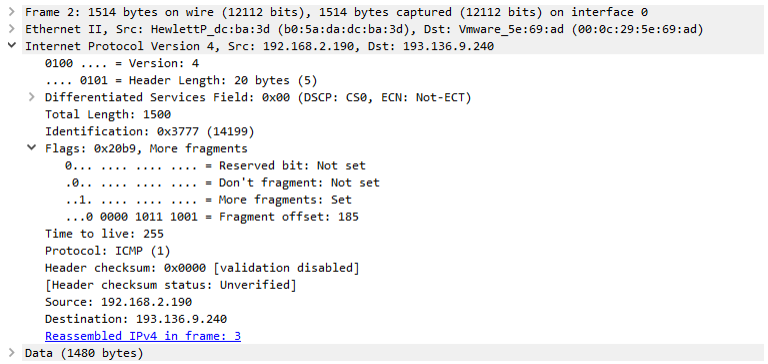

Figura 13: Fragmento 2

4.4 Questão D

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

Foram criados 3 fragmentos, como se pode observar na figura 14. Para detetar o último fragmento do datagrama original procurou-se pelo fragmento no qual o valor da *flag More Fragments* estava definido como *Not set*.

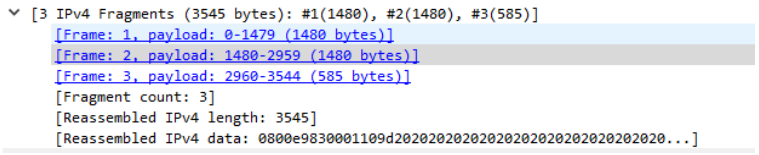

Figura 14: Fragmentos criados

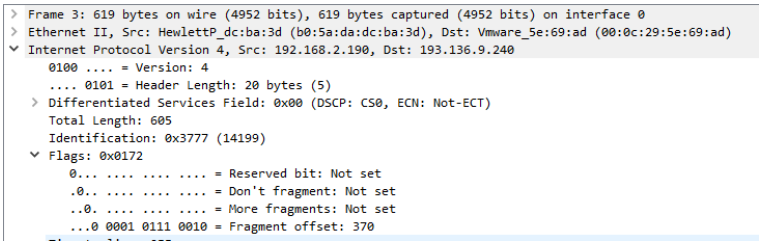

Figura 15: Último fragmento

4.5 Questão 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."

Tanto o primeiro como o segundo fragmento têm a mesma *flag More Fragments* assinalada a 1 e com o valor *Set*. Também o tamanho destes dois fragmentos é igual, 1500 bytes, que é o tamanho da *Maximum Transmission Unit* (MTU), valor que não pode ser excedido. No terceiro fragmento a *flag More Fragments* deixa de estar assinalada a 1, passando a 0, e deixa também de ter o valor *Set*, ficando com *Not Set*. Aqui, o tamanho deixa de ser 1500 bytes pois a informação deste fragmento é inferior à MTU. O campo que muda nos três fragmentos é o campo *Fragment offset* e é este que nos permite reconstruir o datagrama original juntando todos os fragmentos.

5 Grupo I - Parte 2

5.1 Questão A

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

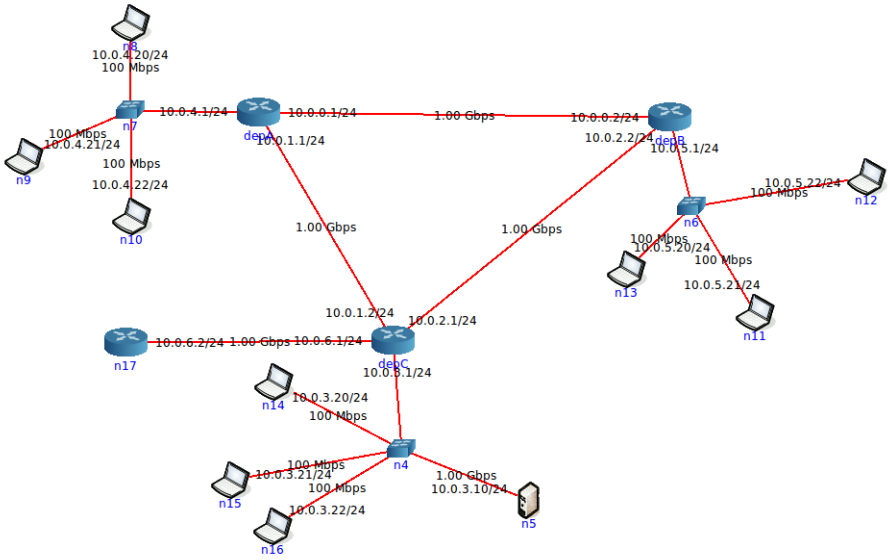

Figura 16: Topologia da rede

A máscara de rede utilizada é 255.255.255.0 uma vez que os *routers* da rede e os hosts da rede têm endereços /24 atribuídos, ou seja, nestes endereços, 24 bits dos 32 totais identificam a rede.

5.2 Questão B

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

Em redes, o seguinte conjunto de endereços corresponde a endereços privados:

- 10.0.0.0 até 10.255.255.255
- 172.16.0.0 até 172.31.255.255
- 192.168.0.0 até 192.168.255.255
- 169.254.0.0 até 169.254.255.255

Por observação da topologia e dos endereços atribuídos pelo CORE e com o conhecimento acima descrito, é possível afirmar que trata-se de endereços privados.

5.3 Questão C

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

Os *switches* não operam ao nível de rede e, por isso, a camada de rede (neste caso, o IP), é completamente transparente para este tipo de dispositivos. Fazem apenas *forward* do pacote para o host destino.

5.4 Questão D

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

```
root@n8:/tmp/pycore.33929/n8.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=62 time=0.044 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=62 time=0.059 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=62 time=0.044 ms
64 bytes from 10.0.3.10: icmp_req=4 ttl=62 time=0.165 ms
^C
--- 10.0.3.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.044/0.078/0.165/0.050 ms
```

Figura 17: Teste de conectividade entre departamento A e o servidor

```
root@n12:/tmp/pycore.33929/n12.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=62 time=0.098 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=62 time=0.162 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=62 time=0.134 ms
64 bytes from 10.0.3.10: icmp_req=4 ttl=62 time=0.141 ms
^C
--- 10.0.3.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.098/0.133/0.162/0.027 ms
```

Figura 18: Teste de conectividade entre departamento B e o servidor

```
root@n14:/tmp/pycore.33929/n14.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=64 time=0.064 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=64 time=0.103 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=64 time=0.106 ms
64 bytes from 10.0.3.10: icmp_req=4 ttl=64 time=0.103 ms
^C
--- 10.0.3.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.064/0.094/0.106/0.017 ms
```

Figura 19: Teste de conectividade entre departamento C e o servidor

5.5 Questão E

"Verifique se existe conectividade IP do router de acesso Rex para o servidor S1"

```
root@n17:/tmp/pycore.33929/n17.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=63 time=0.086 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=63 time=0.142 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=63 time=0.153 ms
64 bytes from 10.0.3.10: icmp_req=4 ttl=63 time=0.163 ms
^C
--- 10.0.3.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.086/0.136/0.163/0.029 ms
```

Figura 20: Teste de conectividade entre *router* exterior e o servidor

6 Grupo II - Parte 2

6.1 Questão 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)."

Por análise da figura 21 (tabela de encaminhamento de um host), verifica-se o seguinte:

1. A primeira linha representa a rota por defeito (0.0.0.0). Significa que, todos os datagramas que não fazem *match* com nenhum IP da coluna *Destination*, devem ir para o router de acesso. Por isso, daí em diante, reencaminhamento fica a cargo do *router*.
2. A segunda linha trata de fazer encaminhar pacotes que sejam para a rede local. Ou seja, o próprio host trata de entregar os datagramas aos hosts que estão na mesma sub-rede, uma vez que conhece a topologia.

```
root@n8:/tmp/pycore.33929/n8.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.4.1        0.0.0.0         UG        0 0          0 eth0
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

Figura 21: Tabela de encaminhamento do host n8

Por análise da figura 22 (tabela de encaminhamento de um *router* de acesso), verifica-se que não há rotas por defeito já que todas as rotas existentes na rede devem-se encontrar identificadas na tabela. Então, o *router* deve ser capaz de encaminhar tráfego para as redes 10.0.0.0, 10.0.1.0 e 10.0.4.0, já que em todas as linhas, o *gateway* está com o valor 0.0.0.0. Todas as restantes linhas sugerem as redes às quais o *router* consegue chegar, mas às quais não está diretamente ligado. Por isso, deve reencaminhar os pacotes para os *routers* que o *routing* dinâmico determinou e daí a presença da *flag* G no campo *Flags*.

```
root@depA:/tmp/pycore.33929/depA.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
10.0.1.0        0.0.0.0         255.255.255.0   U         0 0          0 eth1
10.0.2.0        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.3.0        10.0.1.2        255.255.255.0   UG        0 0          0 eth1
10.0.4.0        0.0.0.0         255.255.255.0   U         0 0          0 eth2
10.0.5.0        10.0.0.2        255.255.255.0   UG        0 0          0 eth0
10.0.6.0        10.0.1.2        255.255.255.0   UG        0 0          0 eth1
```

Figura 22: Tabela de encaminhamento do *router* A

6.2 Questão B

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

A partir da imagem abaixo, é possível verificar que está a ser usado o protocolo de encaminhamento OSPF.

OSPF significa *Open Shortest Path First* e é um protocolo de encaminhamento dinâmico que foi criado para substituir o antigo RIP.

Conclui-se, por isso, que está a ser usado encaminhamento dinâmico e, como tal, as rotas são calculadas automaticamente.

```
depB# show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP,
       O - OSPF, o - OSPF6, I - IS-IS, B - BGP, A - Babel,
       > - selected route, * - FIB route

O   10.0.0.0/24 [110/10] is directly connected, eth0, 00:11:59
C>* 10.0.0.0/24 is directly connected, eth0
O>* 10.0.1.0/24 [110/20] via 10.0.0.1, eth0, 00:11:14
  *                     via 10.0.2.1, eth1, 00:11:14
O   10.0.2.0/24 [110/10] is directly connected, eth1, 00:11:59
C>* 10.0.2.0/24 is directly connected, eth1
O>* 10.0.3.0/24 [110/20] via 10.0.2.1, eth1, 00:11:14
O>* 10.0.4.0/24 [110/20] via 10.0.0.1, eth0, 00:11:14
O   10.0.5.0/24 [110/10] is directly connected, eth2, 00:11:59
C>* 10.0.5.0/24 is directly connected, eth2
O>* 10.0.6.0/24 [110/20] via 10.0.2.1, eth1, 00:11:14
C>* 127.0.0.0/8 is directly connected, lo
```

Figura 23: Informações de encaminhamento no *router* B

6.3 Questão 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 C. Use o comando route delete para o efeito. Que implicações tem esta medida para os utilizadores da empresa que acedem ao servidor. Justifique."

```
root@n5:/tmp/pycore.33929/n5.conf# route del default
root@n5:/tmp/pycore.33929/n5.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.3.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

Figura 24: Tabela de encaminhamento após o comando *route del*

Após a remoção da rota *default* ou 0.0.0.0, o *router* não sabe o que fazer aos pacotes que não fazem *match* com o que está na tabela de encaminhamento. No entanto, não se removeu da tabela o encaminhamento para a rede local e, por isso, os utilizadores do departamento continuam com conectividade, como se mostra na imagem abaixo. Em relação a pacotes externos para o *router*, este consegue recebê-los, mas não consegue responder.

```
root@n14:/tmp/pycore.33929/n14.conf# ping 10.0.3.10
PING 10.0.3.10 (10.0.3.10) 56(84) bytes of data.
64 bytes from 10.0.3.10: icmp_req=1 ttl=64 time=0.067 ms
64 bytes from 10.0.3.10: icmp_req=2 ttl=64 time=0.034 ms
64 bytes from 10.0.3.10: icmp_req=3 ttl=64 time=0.091 ms
64 bytes from 10.0.3.10: icmp_req=4 ttl=64 time=0.130 ms
^C
--- 10.0.3.10 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2997ms
rtt min/avg/max/mdev = 0.034/0.080/0.130/0.036 ms
```

Figura 25: Teste de conectividade para a rede local

6.4 Questão D e Questão E

"Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor S1, por forma a contornar a restrição imposta na alínea c). Utilize para o efeito o comando route add e registe os comandos que usou. Teste a nova política de encaminhamento garantindo que o servidor está novamente acessível, utilizando para o efeito o comando ping. Registe a nova tabela de encaminhamento do servidor. "

Foram adicionadas rotas estáticas para reestabelecer a conexão entre os departamentos, estabelecendo-se que, para a rede local seria o próprio servidor a encaminhar. Para as restantes redes da topologia, seria o *router* de acesso o responsável pelo encaminhamento.

Na figura 26, é possível ver os comandos usados para o efeito.

```
root@n5:/tmp/pycore.33929/n5.conf# route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.3.1
root@n5:/tmp/pycore.33929/n5.conf# route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.3.1
root@n5:/tmp/pycore.33929/n5.conf# route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.3.1
root@n5:/tmp/pycore.33929/n5.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.3.1
root@n5:/tmp/pycore.33929/n5.conf# route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.3.1
root@n5:/tmp/pycore.33929/n5.conf# route add -net 10.0.6.0 netmask 255.255.255.0 gw 10.0.3.1
root@n5:/tmp/pycore.33929/n5.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth0
10.0.1.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth0
10.0.2.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth0
10.0.3.0        10.0.3.10       255.255.255.0   UG        0 0          0 eth0
10.0.4.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth0
10.0.5.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth0
10.0.6.0        10.0.3.1        255.255.255.0   UG        0 0          0 eth0
```

Figura 26: Adição de rotas estáticas e nova tabela de encaminhamento

```
root@n5:/tmp/pycore.33929/n5.conf# ping 10.0.4.20
PING 10.0.4.20 (10.0.4.20) 56(84) bytes of data.
64 bytes from 10.0.4.20: icmp_req=1 ttl=62 time=0.048 ms
64 bytes from 10.0.4.20: icmp_req=2 ttl=62 time=0.118 ms
64 bytes from 10.0.4.20: icmp_req=3 ttl=62 time=0.114 ms
64 bytes from 10.0.4.20: icmp_req=4 ttl=62 time=0.055 ms
^C
--- 10.0.4.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.048/0.083/0.118/0.034 ms
```

Figura 27: Teste de conectividade com rota estática

7 Grupo III - Parte 2

7.1 Questão A

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

Para a atribuição de endereços às redes dos departamentos, decidiu-se fazer subnetting. Dos cálculos efetuados, a partir do endereço IP 172.65.48.0/20, surgiram 6 sub-redes uma vez que se usaram 3 bits para sub-redes sobrando 9 bits para hosts (12 é total de bits nos quais podemos mexer).

É importante referir que não era possível usar somente 2 bits porque 2^2 = 4 e 4 - 2 = 2, que é menor que 3.

Portanto, usando 3 bits e passando para /23, manifestaram-se 8 endereços:

- 172.65.48.0/23 - DESCARTADO
- 172.65.50.0/23 - Sub-Rede 1
- 172.65.52.0/23 - Sub-Rede 2
- 172.65.54.0/23 - Sub-Rede 3
- 172.65.56.0/23 - Sub-Rede 4
- 172.65.58.0/23 - Sub-Rede 5
- 172.65.60.0/23 - Sub-Rede 6
- 172.65.62.0/23 - DESCARTADO

Usando então as primeiras 3 sub-redes (510 hosts em cada uma), definiu-se a seguinte topologia:

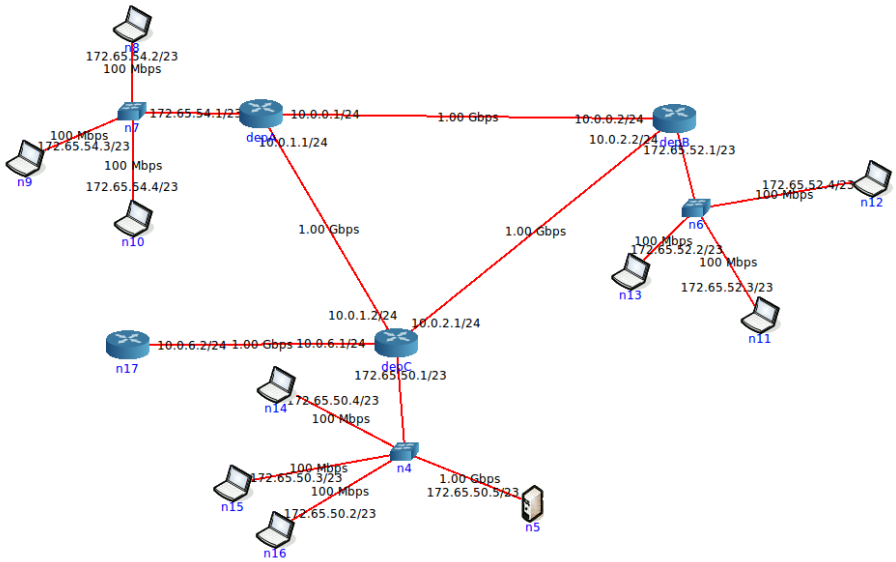

Figura 28: Topologia da rede com *subnetting*

Contudo, verifica-se nesta implementação que há um enorme desaproveitamento de espaço, já que se está a reservar espaço para sub-redes que nunca vão ser usadas no contexto do problema. Por isso, optou-se por fazer *supernetting*.

Surgiu daí 3 endereços:

- 172.65.50.0/23 e 172.65.52.0/23 resulta em 172.65.48.0/22
- 172.65.54.0/23 e 172.65.56.0/23 resulta em 172.65.52.0/22
- 172.65.58.0/23 e 172.65.60.0/23 resulta em 172.65.56.0/22

Ou seja, a partir do *supernetting* agregando dois a dois, conseguimos ter 1022 hosts em cada sub-rede em vez dos 510 hosts iniciais.

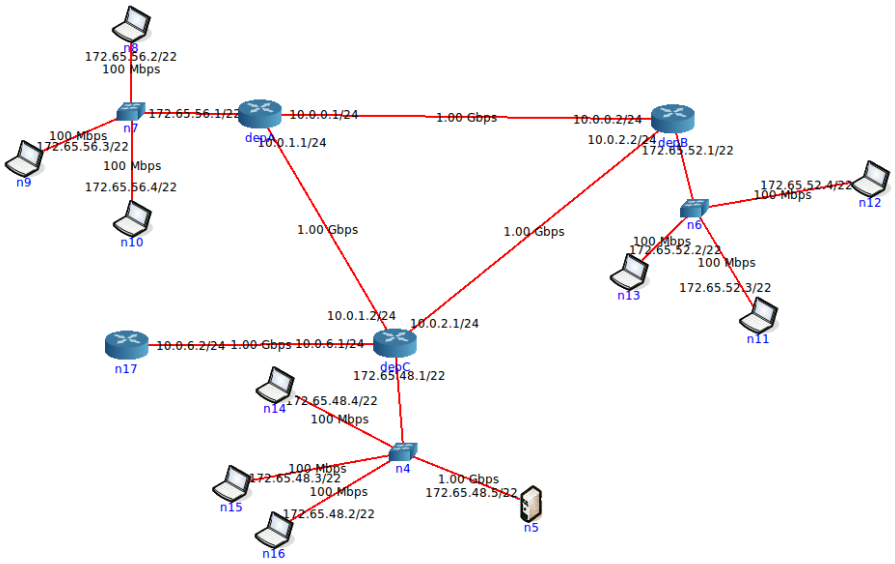

Figura 29: Topologia da rede com *supernetting*

7.2 Questão B

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

A partir do *subnetting* inicial, o grupo usou um /23 que corresponde a 255.255.254.0 e, no qual, se pode interligar 510 hosts em cada departamento. Depois, verificou-se que havia um desaproveitamento do espaço de endereçamento e, com base em *supernetting*, passou-se a usar um /22 que corresponde a 255.255.252.0 e, no qual, se pode interligar 1022 hosts em cada departamento.

7.3 Questão C

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

Verifica-se há conectividade entre as várias redes locais da organização.

```
root@n5:/tmp/pycore.53797/n5.conf# ping 172.65.56.2
PING 172.65.56.2 (172.65.56.2) 56(84) bytes of data.
64 bytes from 172.65.56.2: icmp_req=1 ttl=62 time=0.121 ms
64 bytes from 172.65.56.2: icmp_req=2 ttl=62 time=0.119 ms
^C
--- 172.65.56.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.119/0.120/0.121/0.001 ms
```

Figura 30: Teste de conectividade entre departamento C e A

```
root@n5:/tmp/pycore.53797/n5.conf# ping 172.65.52.4
PING 172.65.52.4 (172.65.52.4) 56(84) bytes of data.
64 bytes from 172.65.52.4: icmp_req=1 ttl=62 time=0.114 ms
64 bytes from 172.65.52.4: icmp_req=2 ttl=62 time=0.630 ms
^C
--- 172.65.52.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.114/0.372/0.630/0.258 ms
```

Figura 31: Teste de conectividade entre departamento C e B

```
root@n11:/tmp/pycore.53797/n11.conf# ping 172.65.56.4
PING 172.65.56.4 (172.65.56.4) 56(84) bytes of data.
64 bytes from 172.65.56.4: icmp_req=1 ttl=62 time=0.112 ms
64 bytes from 172.65.56.4: icmp_req=2 ttl=62 time=0.119 ms
^C
--- 172.65.56.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.112/0.115/0.119/0.011 ms
```

Figura 32: Teste de conectividade entre departamento B e A

8 Conclusão

Neste trabalho aprofundou-se a compreensão do funcionamento interno das redes IP, principalmente nas suas estruturas e na transmissão de dados. Em particular, estudou-se o *subnetting*, *supernetting* e as suas aplicações bem como o potencial do *routing* dinâmico e estático e a importância das tabelas de endereçamento. Em relação aos datagramas, notou-se a importância do TTL, dos diferentes protocolos IP e informações presentes tanto em *headers* como em *payloads*.