

# **Redes de Computadores**

## **Ensaio - 2º Trabalho Prático | Protocolo IP**

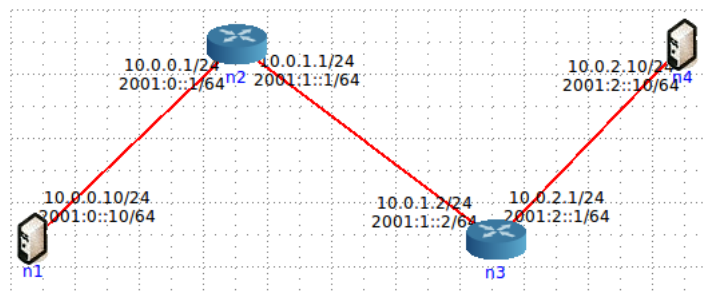
Paulo Caldas (a79089), Pedro Henrique (a77377), Vitor Peixoto (a79175)

Universidade do Minho, Departamento de Informática, 4710-057 Braga, Portugal  
e-mail: {a79089, a77377, a79175}@alunos.uminho.pt

## Parte 1 - Datagramas IP e Fragmentação

### 1

O primeiro passo passará por construir a topologia CORE, com os *hosts* n1 e n4 e os *routers* n2 e n3, estando o *host* n1 ligado ao *router* n2, n2 ligado ao *router* n3, que por sua vez se encontra ligado ao *host* n4, tal como é indicado no enunciado do problema. Finalmente, é iniciada a sessão do modelo de rede associado a esta topologia.



**Figura 1.** Topologia CORE

#### a)

Recorrendo ao comando *tcpdump*, procuramos obter um extrato dos pacotes com origem no *host* n1, assim como outro extrato dos pacotes que têm n1 como destino, guardando os *outputs* resultantes destas filtragens em dois ficheiros distintos, com o nome *source* e *destination*, respetivamente.

Os comandos utilizados foram:

```
tcpdump -i eth0 src 10.0.0.10 -w source
tcpdump -i eth0 dst 10.0.0.10 -w destination
```

De seguida, é executado o comando *traceroute* a partir de n1 e com o endereço IP de destino correspondente ao do *host* n4, que permitirá descobrir a rota IP da origem ao destino.

O comando utilizado é o que se segue:

```
traceroute -I 10.0.2.10
```

#### b)

Possuímos agora dois ficheiros para analisar. Usando o *Wireshark*, são agora abertos os ficheiros que contêm os extratos dos pacotes com origem e destino no *host* n1. Com recurso ao filtro *icmp*, obtemos apenas conteúdo relativo ao protocolo ICMP (*Internet Control Message Protocol*).

Analisando o ficheiro *source*, é possível verificar que o tráfego com origem no *host* n1 (IP 10.0.0.10) consiste em vários *ping requests*, com destino ao *host* n4 (IP 10.0.2.10), com TTL (*time-to-live*) crescente. Inicialmente 1, é incrementado em 1

unidade a cada 3 pacotes, como é expectável, uma vez que tal como é descrito no enunciado, o comando *traceroute* faz com que sejam enviados 3 datagramas (correspondentes aos previamente mencionados *ping requests*) com TTL inicial de 1, para dessa forma identificar o elemento da rede a 1 salto de distância, recorrendo a uma mensagem ICMP a ser enviada por este último à origem, neste caso, o *host* n1. Isto ocorre sucessivamente, com TTL crescente, até ser atingido o elemento da rede com o dado IP de destino, neste caso, o *host* n4. Do comando *traceroute* resultam os endereços IP dos elementos da rede a cada salto até ao destino e por cada uma destas entradas são apresentados 3 *ping times*, calculados com o envio de cada um dos 3 pacotes a cada IP.

```

root@n1:/tmp/pycore.60616/n1.conf# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1  10.0.0.1 (10.0.0.1)  0.071 ms  0.010 ms  0.005 ms
 2  10.0.1.2 (10.0.1.2)  0.031 ms  0.009 ms  0.007 ms
 3  10.0.2.10 (10.0.2.10)  0.056 ms  0.014 ms  0.010 ms
root@n1:/tmp/pycore.60616/n1.conf#

```

**Figura 2.** Output resultante do *traceroute*

ICMP	74	Echo (ping) request	id=0x0043, seq=1/256, ttl=1 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=2/512, ttl=1 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=3/768, ttl=1 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=4/1024, ttl=2 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=5/1280, ttl=2 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=6/1536, ttl=2 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=7/1792, ttl=3 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=8/2048, ttl=3 (no response found!)
ICMP	74	Echo (ping) request	id=0x0043, seq=9/2304, ttl=3 (no response found!)

**Figura 3.** Demonstração de TTL a ser incrementado a cada três pacotes

Relativamente ao ficheiro *destination*, que contém o extrato dos pacotes recebidos pelo *host* n1 (IP 10.0.0.10), é possível distinguir 2 tipos de mensagens ICMP diferentes: 6 delas apresentam a informação *time-to-live exceeded in transit* - 3 enviadas pelo *router* n2 (IP 10.0.0.1) e outras 3 pelo *router* n3 (IP 10.0.1.2); as restantes correspondem a *ping replies* enviados pelo *host* n4 (IP 10.0.2.10). Este resultado era também expectável, pois os datagramas com TTL de 1 ou 2 não chegarão a n4, o destino, o que provocará as mensagens que verificamos que n2 e n3 enviam. Para valores de TTL maiores ou iguais a 3, os datagramas chegarão ao destino e verificamos então a ocorrência de *ping replies* enviados por n4. O TTL destas mensagens será 62, o que corresponde ao valor por defeito, 64, com 2 decrementos de 1 unidade, que ocorrem no processamento dos pacotes nos *routers* n2 e n3.

Source	Destination	Protocol	Length	Info
10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
10.0.0.1	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
10.0.1.2	10.0.0.10	ICMP	102	Time-to-live exceeded (Time to live exceeded in transit)
10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0045, seq=7/1792, ttl=62
10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0045, seq=8/2048, ttl=62
10.0.2.10	10.0.0.10	ICMP	74	Echo (ping) reply id=0x0045, seq=9/2304, ttl=62

**Figura 4.** Pacotes recebidos por n1: *ping replies* e *TTL exceeded*

c)

O valor inicial mínimo do campo TTL para alcançar o destino n4 deverá ser 3. Se for 1 ou 2, o pacote será descartado pelo primeiro ou segundo *routers* e prontamente será enviada a mensagem ICMP informando a origem que o *time-to-live* foi excedido em trânsito.

Para efeitos de verificação, com alguma redundância, pois apenas confirmam os resultados obtidos na alínea anterior, apoiados pela topologia criada, executamos o comando *ping*, com origem em n1 e destino em n4 e valores de TTL variados, e de seguida serão analisados os resultados.

O comando utilizado é então:

*ping -t x -c 1 10.0.2.10*

A *flag -c* indica o número de pacotes a serem enviados ao endereço IP de destino (neste caso, 1). A *flag -t* indica o valor do campo TTL do pacote.

E os resultados da execução são:

```

vcmd
root@n1:/tmp/pycore.33100/n1.conf# ping -t 1 -c 1 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
From 10.0.0.1 icmp_seq=1 Time to live exceeded

--- 10.0.2.10 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@n1:/tmp/pycore.33100/n1.conf# ping -t 2 -c 1 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
From 10.0.1.2 icmp_seq=1 Time to live exceeded

--- 10.0.2.10 ping statistics ---
1 packets transmitted, 0 received, +1 errors, 100% packet loss, time 0ms

root@n1:/tmp/pycore.33100/n1.conf# ping -t 3 -c 1 10.0.2.10
PING 10.0.2.10 (10.0.2.10) 56(84) bytes of data.
64 bytes from 10.0.2.10: icmp_req=1 ttl=62 time=0.039 ms

--- 10.0.2.10 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.039/0.039/0.039/0.000 ms
root@n1:/tmp/pycore.33100/n1.conf#

```

**Figura 5.** Comandos *ping* a serem executados com diferentes valores de TTL

Isto confirma que, de facto, o valor do campo TTL necessita de ser **maior ou igual a 3** de modo a que o pacote não seja descartado e chegue então ao *host* n4.

d)

Para determinar o tempo de ida-e-volta médio, serão utilizados os valores apresentados pelo output do comando *traceroute* de n1 para n4. Este apresenta os três tempos (*ping times*) obtidos para cada salto até ao destino. Referente ao tempo de ida-e-volta do *host* n1 para o *host* n4, temos:

$$tempo\_medio = \frac{0.056+0.014+0.010}{3}ms = 0.02(6)ms$$

2

Começamos por executar o comando:

*traceroute -I router - di.uminho.pt*

a)

O endereço IP da interface ativa do computador utilizado para fazer este trabalho pode ser determinado recorrendo às primeiras mensagens ICMP enviadas, associadas a *ping requests* resultantes da execução do comando *traceroute*, identificando o IP de origem: 192.168.100.159. Outra indicação do IP de origem é o facto de apenas um IP receber mensagens *TTL exceeded*, ou seja, foi da interface identificada por esse endereço IP que o tráfego gerado pelo comando *traceroute* partiu.

No.	Time	Source	Destination	Protocol	Length	Info
130	5.026690414	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=1/256, ttl=1 (no response found!)
131	5.026700450	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=2/512, ttl=1 (no response found!)
132	5.026704427	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=3/768, ttl=1 (no response found!)
133	5.026708521	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=4/1024, ttl=2 (reply in 158)
134	5.026711871	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=5/1280, ttl=2 (reply in 159)
135	5.026715114	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=6/1536, ttl=2 (reply in 160)
136	5.026718778	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=7/1792, ttl=3 (reply in 161)
137	5.026722044	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=8/2048, ttl=3 (reply in 162)
138	5.026725484	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=9/2304, ttl=3 (reply in 163)
139	5.026729193	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=10/2560, ttl=4 (reply in 164)
140	5.026732774	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=11/2816, ttl=4 (reply in 165)
141	5.026736726	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=12/3072, ttl=4 (reply in 166)
142	5.026740450	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=13/3328, ttl=5 (reply in 167)
143	5.026743884	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=14/3584, ttl=5 (reply in 168)
144	5.026746950	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=15/3840, ttl=5 (reply in 169)
145	5.026750557	192.168.100.159	193.136.9.254	ICMP	76	Echo (ping) request id=0x5181, seq=16/4096, ttl=6 (reply in 170)
146	5.027208595	192.168.100.254	192.168.100.159	ICMP	104	Time-to-live exceeded (Time to live exceeded in transit)
147	5.027230415	192.168.100.254	192.168.100.159	ICMP	104	Time-to-live exceeded (Time to live exceeded in transit)
148	5.027234070	192.168.100.254	192.168.100.159	ICMP	104	Time-to-live exceeded (Time to live exceeded in transit)

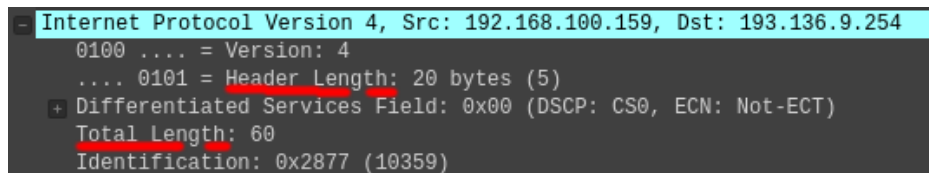
**Figura 6.** Tráfego gerado pelo comando *traceroute*

b)

O valor do campo protocolo é ICMP (1). O *Internet Control Message Protocol* é utilizado para a comunicação entre dispositivos no nível de rede, com funções de controlo (como reportar erros, por exemplo). Geralmente não possui *payload* útil, o que significa que o protocolo ICMP não é utilizado para transferência de dados entre sistemas.

c)

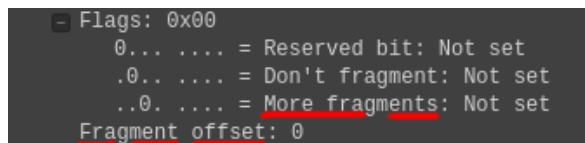
Analisando o primeiro pacote ICMP capturado pelo *Wireshark*, podemos verificar que o tamanho do cabeçalho IPv4 é de 20 *bytes*. O tamanho do *payload* corresponderá ao tamanho total do pacote menos o tamanho do cabeçalho, ou seja,  $60 - 20 = 40$  *bytes*.



**Figura 7.** Tamanho do cabeçalho e tamanho total do pacote

d)

Para determinar se o datagrama IP foi ou não fragmentado, devemos proceder a verificar o valor da *flag more fragments* e do campo *fragment offset*. Como é possível verificar pela imagem, estes encontram-se a 0, o que indica que o datagrama não se encontra fragmentado.



**Figura 8.** *Flag more fragments* e campo *fragment offset*

e)

Analisando os pacotes enviados pela interface ativa do computador, os campos do cabeçalho IP que variam são:

- o TTL (*time-to-live*), que é incrementado a cada três pacotes;
- o ID único de cada pacote;
- o *header checksum*.

f)

O campo de identificação é único para cada pacote e é incrementado em 1 unidade por cada pacote criado, enquanto que o TTL é igual entre cada conjunto de 3 pacotes. Isto acontece porque o programa *traceroute*, partindo de um valor inicial, incrementa o valor do TTL em 1 unidade por cada envio de um grupo de 3 pacotes. Dessa forma obtém 3 tempos distintos de ida-e-volta e assim passa a possuir alguma informação redundante que pode ser útil em casos em que o envio de um pacote falhe pontualmente por alguma razão arbitrária, sendo agora possível obter um *ping time* no envio dos restantes pacotes com o mesmo TTL, ou até mesmo para poder criar uma melhor percepção sobre o *ping time* verdadeiro relativo a um pacote com um determinado TTL, pois assim qualquer atraso pontual pode ser identificado em comparação com os tempos obtidos para os restantes pacotes de igual TTL.

g)

O campo TTL dos pacotes relativos à série de respostas ICMP TTL *exceeded* é igual entre si e toma o valor de 64. Este valor permanece constante para todas as mensagens de resposta ICMP TTL *exceeded* enviadas ao nosso *host* pois é um valor por defeito que provém e é definido pelo próprio *router* de destino.

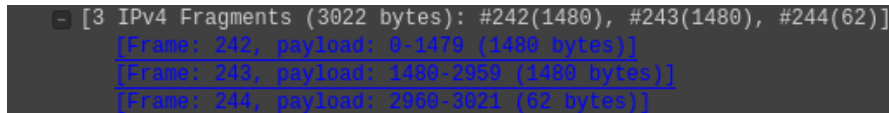
### 3

Começamos por executar o comando *traceroute* utilizando pacotes de 3042 *bytes*:

*traceroute -I 3042*

a)

Após localizar a primeira mensagem ICMP verificamos que, de facto, esta foi fragmentada. Houve a necessidade de o fazer, uma vez que a MTU (*maximum transmission unit*) de pelo menos um dos *routers* é menor do que o tamanho do pacote inicial enviado (3042 *bytes*).



```
[3 IPv4 Fragments (3022 bytes): #242(1480), #243(1480), #244(62)]
[Frame: 242, payload: 8-1479 (1480 bytes)]
[Frame: 243, payload: 1480-2959 (1480 bytes)]
[Frame: 244, payload: 2960-3021 (62 bytes)]
```

**Figura 9.** Demonstração dos 3 fragmentos usados para reconstruir o primeiro pacote.

b)

É possível identificar que o datagrama foi fragmentado recorrendo à *flag more fragments*, apresentando esta o valor 1. Sabemos que se trata do primeiro fragmento pois o valor que o campo *fragment offset* toma é 0. O tamanho deste datagrama IP é de 1500 *bytes* (20 de cabeçalho + 1480 de dados), que corresponde à MTU de pelo menos um dos *routers* que encaminhou o tráfego gerado pelo comando *traceroute*.

c)

Podemos afirmar que este fragmento não se trata do primeiro do datagrama IP original já que este apresenta um valor diferente de 0 no campo *fragment offset* do cabeçalho. Podemos também inferir a existência de fragmentos adicionais pela *flag more fragments*, que toma o valor 1.

d)

Foram criados 3 fragmentos, que partilham o mesmo valor de identificação no cabeçalho IP, relativo ao datagrama IP original. O último fragmento é detetado pela *flag more fragments*, que toma o valor 0, e o campo *fragment offset*, quando este toma um valor diferente de 0. Isto significa que o fragmento sob análise corresponde ao último referente a um dado datagrama IP original.

e)

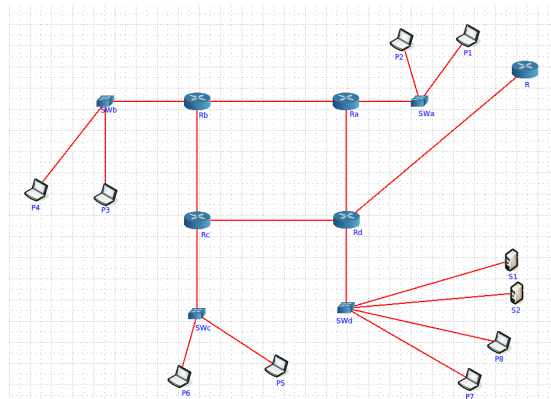
Os campos que mudam no cabeçalho IP entre os diferentes fragmentos são a *flag* que informa sobre a fragmentação de um pacote e o campo *fragment offset*. Para a reconstrução do datagrama IP original, é utilizado o seu identificador, presente em todos os seus fragmentos, que permite assim associá-los. Para além disto, temos as *flags* relativas à fragmentação e o valor do campo *fragment offset*, sendo estes dados úteis para determinar a possível ocorrência de fragmentação de um pacote. O valor da *flag* poderá indicar a existência de *more fragments* (MF) para todos os fragmentos de um pacote, excluindo o último. O valor de *fragment offset* permite sequenciar os fragmentos e determinar a posição do fragmento na reconstrução do datagrama original. É também útil para identificar o último fragmento de um dado pacote, uma vez que este indica a não existência de mais fragmentos, sendo MF = 0. A única distinção entre o fragmento e um pacote não fragmentado será o valor do *offset*, que sendo diferente de 0 indica a ocorrência de fragmentação. O campo que indica o tamanho do pacote será também útil na reassemblagem, sendo a partir dele possível perceber quando o datagrama original está finalmente reconstruído.



## Parte 2

### 1

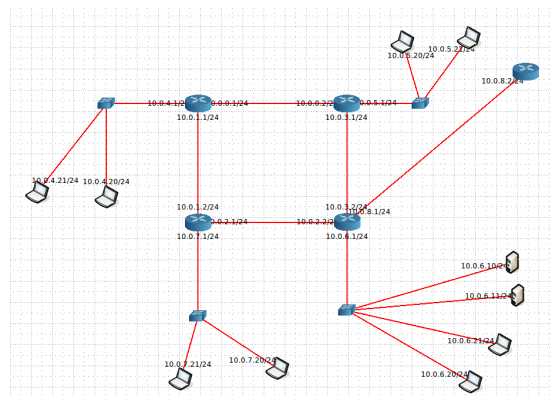
Começamos por recriar a topologia indicada no enunciado do problema:



**Figura 10.** Topologia utilizada e nomes de cada dispositivo

#### a)

Os respectivos endereços IP e máscara de rede atribuídos a cada interface da rede foram os seguintes:



**Figura 11.** Visualização dos endereços IP e máscaras de rede da topologia

A máscara de rede será 11111111.11111111.11111111.00000000 ou 255.255.255.0, que corresponde a /24 em notação CIDR.

b)

Segundo o padrão RFC 1918, os endereços IP que não devem ser utilizados na Internet, mas que no entanto funcionam como endereços privados são os compreendidos nos seguintes intervalos:

192.168.0.0 – 192.168.255.255  
172.16.0.0 – 172.31.255.255  
10.0.0.0 – 10.255.255.255

Assim sendo, podemos concluir que os endereços que vemos na topologia CORE são **privados**.

c)

Os *switches* não lhes têm atribuídos endereços IP, uma vez que estes funcionam no segundo nível do modelo OSI (ligação de dados). As unidades de dados protocolares neste nível são designadas por *frames*, enquanto que na 3<sup>a</sup> camada, a de rede, se designam por datagramas ou pacotes. Isto significa que os *switches* não têm endereços IP associados às suas interfaces, pois não há essa noção no nível a que estes operam, uma vez que o *Internet Protocol* e os seus métodos de endereçamento são referentes ao nível acima.

d)

Para verificar a conectividade entre o servidores e os vários portáteis podíamos correr, na *shell* origem e destino respetivamente, comandos *ping* e *tcpdump*. Por outro lado, o programa CORE permite verificar conectividade entre certos nodos. Seguem-se a visualização de alguns testes:

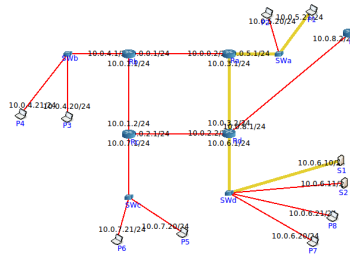


Figura 12. P1 com S1

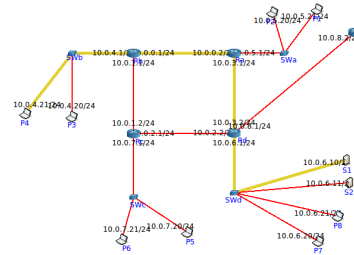


Figura 13. P4 com S1

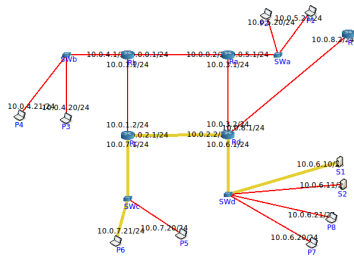


Figura 14. P6 com S1

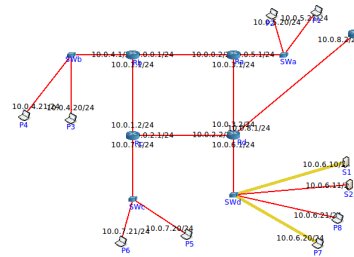


Figura 15. P7 com S1

e)

Utilizando o mesmo método da alínea anterior, verificamos que existe de facto conectividade entre os servidores e o *router* de acesso:

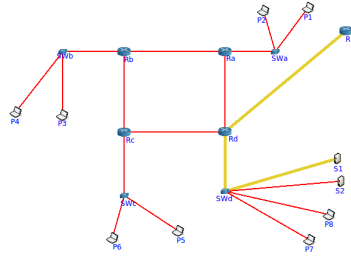


Figura 16. S1 com R

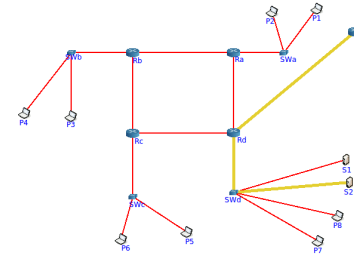


Figura 17. S2 com R

2

a)

Esta é a tabela de endereçamento para o *router* do departamento B:

<i>Destination</i>	<i>Gateway</i>	<i>Genmask</i>	<i>Flags</i>	<i>MSS</i>	<i>Window</i>	<i>irtt</i>	<i>Iface</i>
10.0.0.0	0.0.0.0	255.255.255.0	<i>U</i>	0	0	0	<i>eth0</i>
10.0.1.0	0.0.0.0	255.255.255.0	<i>U</i>	0	0	0	<i>eth1</i>
10.0.2.0	10.0.1.2	255.255.255.0	<i>UG</i>	0	0	0	<i>eth1</i>
10.0.3.0	10.0.0.2	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.4.0	0.0.0.0	255.255.255.0	<i>U</i>	0	0	0	<i>eth2</i>
10.0.5.0	10.0.0.2	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.6.0	10.0.0.2	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.7.0	10.0.1.2	255.255.255.0	<i>UG</i>	0	0	0	<i>eth1</i>
10.0.8.0	10.0.0.2	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>

Para um dos computadores portáteis do departamento B:

<i>Destination</i>	<i>Gateway</i>	<i>Genmask</i>	<i>Flags</i>	<i>MSS</i>	<i>Window</i>	<i>irtt</i>	<i>Iface</i>
0.0.0.0	10.0.4.1	0.0.0.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.4.0	0.0.0.0	255.255.255.0	<i>U</i>	0	0	0	<i>eth0</i>

Estas são tabelas de *routing* que de cada dispositivo para descobrir o próximo salto que um pacote deverá fazer para chegar ao seu destino.

Por exemplo, no *router* B, se o destino pertencer à rede 10.0.5.0 (por exemplo, o portátil P2 com endereço IP 10.0.5.20), sabe que o seu próximo salto é 10.0.0.2 que, como podemos visualizar na topologia, significa reencaminhar o pacote para o *router* A.

Segundo o mesmo raciocínio no portátil da rede B, se o destino for da rede 10.0.4.0 (a rede onde se encontra) saberá que o próximo salto é 0.0.0.0, que significa que o dispositivo de destino estará na mesma rede local onde o portátil se encontra. Todavia, se for qualquer outro tipo de destino que não corresponda ao previamente discutido (porque só existem duas entradas nesta tabela), o próximo salto será 10.0.4.1, ou seja, o *router* B reencaminhará o pacote.

b)

Para verificar que tipo de encaminhamento está a ser utilizado, podemos alterar existência de ligações entre dispositivos e ver como o sistema reage. De facto, executando o comando *traceroute* a partir de P4 e com destino em P1, verificamos que farão parte da rota os *routers* Rb e Ra. Eliminando a conectividade entre estes dois *routers* e executando o mesmo comando, verificamos que uma rota alternativa é utilizada. Isto é indicação de que encaminhamento dinâmico é utilizado.

Uma outra possibilidade passa por executar **ps -e** na *shell* de um dos *routers* utilizados e confirmar que processos estão a correr. É possível verificar que o processo *ospfd* está a correr, que é um *daemon* utilizado pelos *routers* no *routing* dinâmico.

c)

Na *shell* do servidor 1 é executado o comando:

```
route del -net 0.0.0.0
```

Este retira a entrada da tabela para a rota por defeito. As implicações são as seguintes: o servidor pode comunicar com quaisquer dispositivos que estejam na sua rede, mas não consegue comunicar com quaisquer outros, uma vez que o *gateway* 10.0.6.1 (*router* D) foi removido. Estas possíveis implicações foram confirmadas com uso do comando *traceroute* a partir de dispositivos dentro e fora da rede para o servidor 1.

d)

Para restaurar a conectividade para o servidor 1, executamos na *shell* do servidor 1 os seguintes comandos:

```
route add -net 10.0.0.0 netmask 255.255.255.0 gw 10.0.6.1
route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.6.1
route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.6.1
route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.6.1
route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.6.1
route add -net 10.0.5.0 netmask 255.255.255.0 gw 10.0.6.1
route add -net 10.0.7.0 netmask 255.255.255.0 gw 10.0.6.1
route add -net 10.0.8.0 netmask 255.255.255.0 gw 10.0.6.1
```

O servidor 1 poderá agora encaminhar pacotes com destino nas várias sub-redes da topologia criada, sem que seja utilizada a rota por defeito (0.0.0.0), servindo-se do *router* da sua sub-rede.

e)

Segue-se a tabela de *routing* resultante referente a S1.

<i>Destination</i>	<i>Gateway</i>	<i>Genmask</i>	<i>Flags</i>	<i>MSS</i>	<i>Window</i>	<i>irtt</i>	<i>Iface</i>
10.0.0.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.1.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.2.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.3.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.4.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.5.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.6.0	0.0.0.0	255.255.255.0	<i>U</i>	0	0	0	<i>eth0</i>
10.0.7.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>
10.0.8.0	10.0.6.1	255.255.255.0	<i>UG</i>	0	0	0	<i>eth0</i>

**Nota:** Foi utilizada a mesma máscara do *default gateway* que removemos na alínea anterior.

É feito um *ping request* com destino ao servidor 1 a partir de todos os aparelhos da topologia para verificar se a conectividade do sistema foi recuperada.

### 3

1)

Começamos por analisar o endereço IP que nos é dado:

	< -rede   hosts- >		
<i>Address</i> :	10101100 . 00010000	. 00000000 . 00000000	
	(172)	(16)	(0) (0)
<i>Mask</i> :	11111111 . 11111111	. 00000000 . 00000000	
	(255)	(255)	(0) (0)

São dados 16 bits para identificar a rede e outros 16 bits para identificar interfaces dos *hosts*. Podemos servir-nos destes últimos para criar sub-redes segundo as necessidades da topologia.

Neste problema, o endereçamento entre os *routers* permanece inalterado, pelo que as sub-redes a definir com este novo endereçamento serão apenas as dos departamentos A, B, C e D. É necessário garantir, pelo menos, dois *hosts* utilizáveis para três das sub-redes e quatro *hosts* utilizáveis para o departamento D. Dividimos então o nosso endereço em quatro sub-endereços com igual capacidade de endereçamento de *hosts*, pelo que será necessário garantir quatro *hosts* para todas elas, tendo em consideração os dois endereços especiais:

$$\begin{aligned} 2^n &\geq 4 + 2 &<=> \\ 2^n &\geq 6 &<=> \\ n &\geq \log_2 6 &<=> \\ n &\geq 2.584962501 \end{aligned}$$

Precisamos então de pelo menos três bits para identificar *hosts* em cada sub-rede.

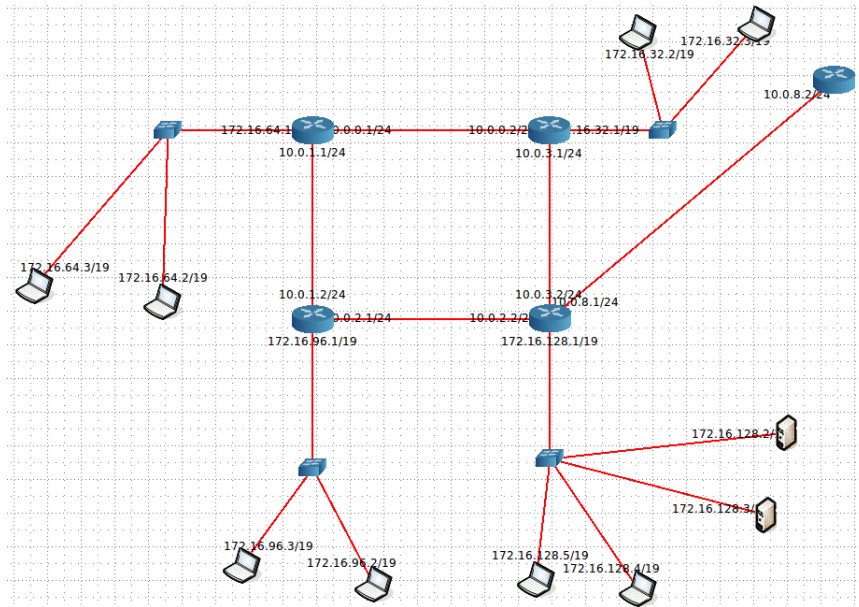
Inicialmente poderíamos pensar em utilizar apenas dois bits endereçar quatro sub-redes, mas necessitamos de reservar 2 endereços para propósitos de identificação da rede global e *broadcasting*. Assim, serão necessários três bits da porção de endereçamento de *hosts* para definir as sub-redes dos departamentos, com um total de  $2^3 = 8$  sub-redes que, não contando com as duas reservadas, nos possibilitam 6 sub-redes utilizáveis.

172.16.0.0/19 : *Reservado*  
 172.16.32.0/19 : *Rede<sub>A</sub>*  
 172.16.64.0/19 : *Rede<sub>B</sub>*  
 172.16.96.0/19 : *Rede<sub>C</sub>*  
 172.16.128.0/19 : *Rede<sub>D</sub>*  
 172.16.160.0/19 : *Livre*  
 172.16.192.0/19 : *Livre*  
 172.16.224.0/19 : *Reservado*

Cada uma das sub-redes possui mais de três bits para identificar *hosts*, pelo que os requisitos são cumpridos.

Todavia, será necessário ter em consideração os riscos e benefícios associados. Por um lado, o uso mínimo de bits para identificação de sub-redes para cumprir os requisitos leva à falta de flexibilidade/adaptabilidade da topologia. Por exemplo, caso sejam criados novos departamentos que exijam conexão à rede, os recursos de endereçamento rapidamente se tornariam escassos e seria necessário reestruturar o endereçamento de forma a cumprir os novos requisitos. Por outro lado, o uso do mínimo número de bits para identificar as redes possibilita o máximo de bits para endereçamento de *hosts*. Isto constitui uma vantagem importante, pois nesta rede será bastante provável haver um grande número de dispositivos conectados, sendo assim útil e benéfico disponibilizar um mais vasto espaço de endereçamento para vários dispositivos.

A topologia final é a seguinte:



**Figura 18.** Topologia utilizando *subnetting*

2)

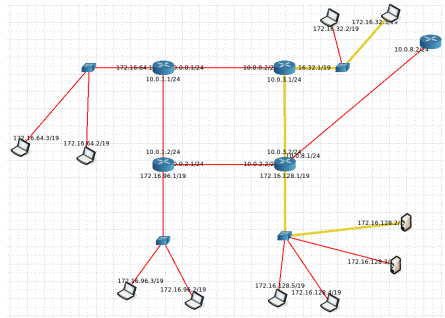
Cada uma das sub redes utiliza 19 bits para identificação de rede. Assim a máscara de rede será:

$$\begin{aligned}
 & \text{Mask : } 11111111 \cdot 11111111 \cdot 111|00000 \cdot 00000000 \\
 & \text{Decimal : (255) \quad (255) \quad (224) \quad (0)}
 \end{aligned}$$

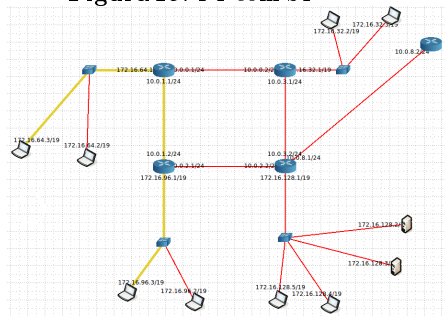
O número de bits dedicados ao endereçamento de *hosts* será 13 e 2 dos endereços totais serão reservados. Assim sendo, cada departamento pode endereçar  $2^{13} - 2 = 8190$  *hosts* IP na sua rede.

**3)**

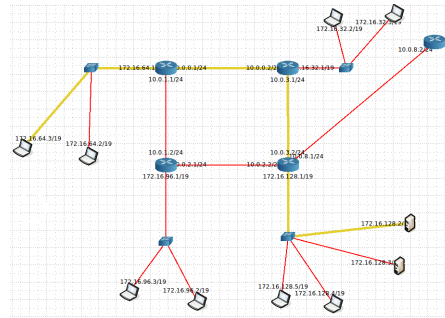
Utilizemos a mesma ferramenta que o CORE disponibiliza para garantir conectividade entre e dentro de departamentos:



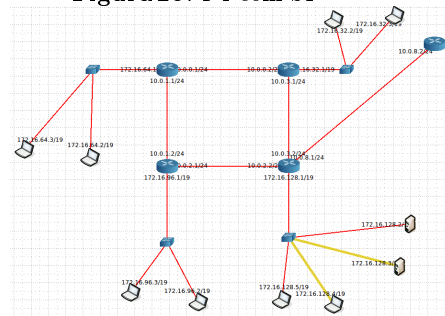
**Figura 19.** P1 com S1



**Figura 21.** P4 com P6



**Figura 20. P4 com S1**



**Figura 22.** P8 com S2