

# **Redes de Computadores**

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

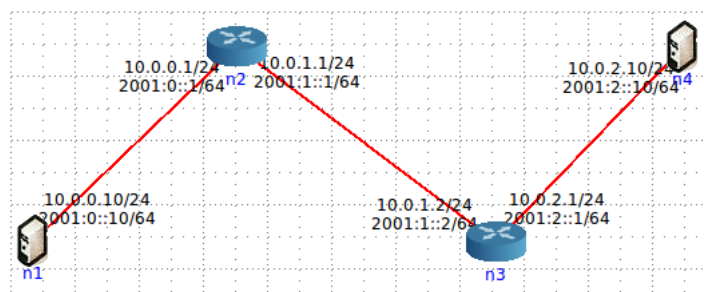
Paulo Caldas (a79089), Pedro Henrique (a77377), Vitor Peixoto (a79175)

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

## Parte 1

### 1

Comecemos por inicializar a topologia CORE, com os host (n1 e n4), os routers (n2 e n3) e as ligações indicadas no problema. Finalmente, inicializamos a sessão do modelo.



**Figura 1.** Topologia CORE

#### a)

Inicializar tcpdump na shell 1: Iremos separar em dois tcpdumps como duas amostras distintas: Na primeira filtramos apenas pacotes em que n1 é origem, e noutro em que é destino, e guardar o registo:

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

Executar traceroute para o IP do host 4: Mais uma vez, executamos este comando (na shell de 1) duas vezes: a primeira para ser captado pelo primeiro tcpdump que guarda em "source", e o segundo que guarda em "destination".

```
traceroute -I 10.0.2.10
```

#### b)

Possuímos agora dois ficheiros para analisar. Como pretendemos analisar apenas um tipo específico de pacote, adicionamos o filtro "ICMP" no Wireshark. Os pacotes resultantes têm todos o mesmo comprimento total.

**Relativamente ao tráfego enviado por 10.0.0.10:** Apenas pings requests para 10.0.2.10 (n4) cujo TTL (time to live) incrementa a cada 3 pacotes

Isto será porque, no comando traceroute, são enviados três pings por TTL. Disto resulta o output de três tempos distintos de modo a dar mais informação sobre o tempo de resposta (como saber a média ou compreender se há uma grande diferença entre valores).

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 2.** Demonstração de TTL a ser incrementado a cada três pacotes

**Relativamente ao tráfego recebido por 10.0.0.10:** Observamos dois tipos de informação: Primeiramente, ping replies enviados por n4. Em segundo lugar, mensagens de "Time-to-live-exceeded". Destes, existem três de n2 e três de n3. Estes são os únicos pontos em que TTL excedido é possível, pois para quaisquer valores maiores existe comunicação possível entre n1 e n4 sem perda de pacote.

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

c)

O valor mínimo do campo TTL para alcançar n4 deverá ser 3. Se for 1, o pacote será descartado pelo primeiro router. Se for 2, será descartado pelo segundo router. Para verificar vamos enviar pings, a partir de n1 para n4, com TTLS variados e verificar o que acontece.

*ping -t x -c 1 10.0.2.10*

A flag -c indica para só ser enviado um pacote ao destino. A flag -t indica qual o valor do campo TTL do pacote. Assim sendo, executamos o ping e avaliamos o resultado no Wireshark:

*Para TTL = 1 : time - to - live exceeded*

*Para TTL = 2 : time - to - live exceeded*

*Para TTL = 3 : ping replied*

Isto confirma que, de facto, **o valor do campo TTL necessita de ser maior ou igual a 3 para o pacote não ser descartado e chegar até n4.**

d)

Para saber o tempo de ida e volta médio, basta ver o output do comando traceroute para n4, que irá mostrar os três tempos obtidos para cada salto. Referente ao tempo de ida e volta para n4, temos:

$$tempoMedio = \frac{0.053+0.031+0.028}{3}ms = 0.0373(3)ms$$

2

Começemos por executar:

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

a)

Para saber o endereço IP da interface ativa do nosso computador, deveremos localizar o primeiro pacote ICMP enviado, e ver qual o IP fonte: **192.168.100.159**. Outra indicação também podia ser o facto de que é o único IP origem que recebe pacotes TTL exceeded (ou seja, foi a partir deste interface que foi executado o traceroute).

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 156)
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	5.026725494	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	5.026746958	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	5.027200595	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 4. Ações principais nos quais a interface ativa do nosso computador participou

b)

O valor do campo protocolo é ICMP: Internet Control Message Protocol. Este é utilizado para controlo a nível de rede e geralmente não possui payload útil, isto é, o protocolo ICMP não é utilizado para trocar dados entre sistemas.

c)

Carregando no primeiro pacote ICMP capturado pelo Wireshark, o tamanho do cabeçalho IPv4 é de 20 bytes. O tamanho do payload será o tamanho total do pacote menos o tamanho do cabeçalho: **60 - 20 = 40 bytes**.

```
- Internet Protocol Version 4, Src: 192.168.100.159, Dst: 193.136.9.254
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
  + Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 60
  Identification: 0x2877 (10359)
```

**Figura 5.** Informações relevantes do pacote usadas para responder a esta pergunta

d)

Saberemos se o datagrama IP não foi fragmentado se a flag "more fragments" estiver a zero e "fragment offset" for zero.

```
- Flags: 0x00
  0... .... = Reserved bit: Not set
  .0.. .... = Don't fragment: Not set
  ..0. .... = More fragments: Not set
  Fragment offset: 0
```

**Figura 6.** Informações relevantes do pacote usadas para responder a esta pergunta

e)

Analisando os pacotes enviados pelo interface ativo do computador, os campos do cabeçalho IP que variam são:

- O TTL, que é incrementado a cada três pacotes
- O ID único de cada pacote

f)

O TTL é incrementado a cada três pacotes, pois o programa traceroute envia três observações para dar três tempos distintos de ida e volta. O campo de identificação é único para cada pacote, e é incrementado por um em cada pacote novo.

g)

A ordem por destino é feita carregando na coluna "destination" do Wireshark. Finalmente, identificamos aquelas com informação de "TTL exceeded". O TTL destes pacotes é o mesmo, 64. É um valor constante para cada pacote pois será definido pelo router e advém sempre da mesma origem(n4).

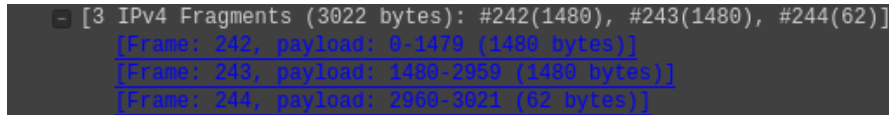
### 3

Comecemos por fazer traceroute por pacotes de 3042 bytes:

*traceroute -I 3042*

a)

Localizando a primeira mensagem ICMP verificamos, de facto, que teve de ser fragmentada. Foi necessário fragmentar pois o MTU (maximum transmission unit) de um dos routers é menor do que o pacote inicial enviado (3042 bytes).



A screenshot of the Wireshark packet list pane. It shows three entries under the heading "[3 IPv4 Fragments (3022 bytes): #242(1480), #243(1480), #244(62)]". Each entry is expanded to show its details: "[Frame: 242, payload: 0-1479 (1480 bytes)]", "[Frame: 243, payload: 1480-2959 (1480 bytes)]", and "[Frame: 244, payload: 2960-3021 (62 bytes)]".

**Figura 7.** Demonstração dos 3 fragmentos usados para reconstruir o primeiro pacote.

b)

Sabemos que o datagrama foi fragmentado pois a flag "more fragments" é um. Sabemos que se trata do primeiro fragmento pois "fragment offset" é zero. O tamanho do datagrama é de 1500 bytes (O MTU de um dos routers que começou a fragmentação).

c)

Estará fragmentado e será o primeiro fragmento se a flag "more fragments" estiver a 1 e o "fragment offset" for igual a zero (sendo portanto o primeiro fragmento do pacote original). Sabemos que haverão mais fragmentos pois "more fragments" é um.

d)

Foram criados 3 fragmentos para compor o pacote inicial. O último fragmento é detectado quando a flag "more fragments" for zero, e o "fragment offset" for diferente de zero. Isto é, o fragmento que está a ser analisado é o último referente a um certo pacote.

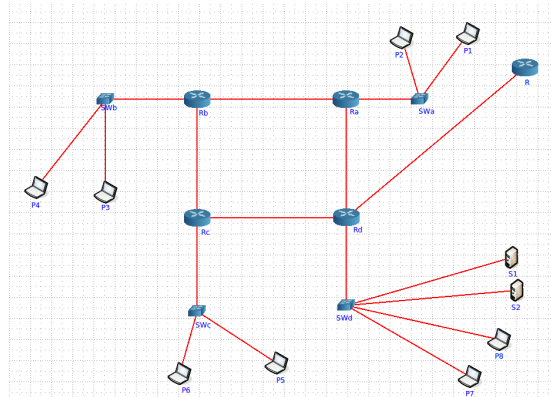
e)

Os campos que mudam são a flag "more fragments" e o ID único de cada pacote. Com este ID conseguimos associar cada fragmento ao pacote inicial ao que pertencia e, com a ajuda do "fragment offset" (outro campo que varia) conseguir construir os fragmentos como pacotes inteiros. Por exemplo, dos pacotes que o Wireshark interceptou, vemos que o pacote 242, 243 e 244 possuem o identificador 0xffdf e os pacotes 245, 246 e 247 possuem o identificador 0xffe0.

## Parte 2

1

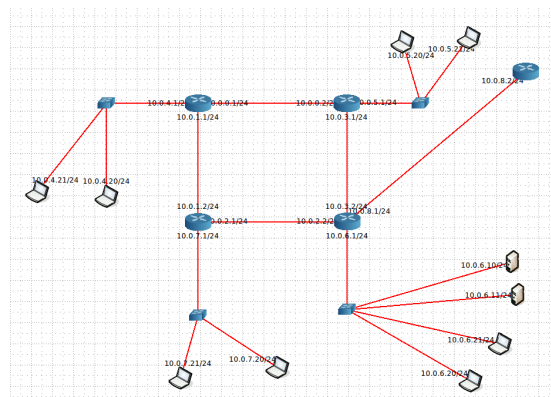
Comecemos por desenhar a topologia indicada no enunciado do problema



**Figura 8.** Topologia utilizada e nomes de cada dispositivo

a)

Os respectivos endereços ip e mascara de rede atribuídos a cada equipamento foram os seguintes:



**Figura 9.** Visualização dos endereços ip e mascaras de rede da topologia

b)

Segundo o padrão RFC 1918, as gamas de endereços IP que não devem ser utilizados na Internet mas funcionam como endereços privados são os seguintes:

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 visualizamos na topologia CORE são **privados**.

c)

Os switches não lhes têm atribuídos endereços IP pois estes funcionam no segundo nível do modelo OSI (Ligação de dados). Elaborando, os switches não necessitam de endereço IP pois **não comunicam utilizando pacotes** (pacotes esses que necessitam de endereços IP para comunicar).

d)

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

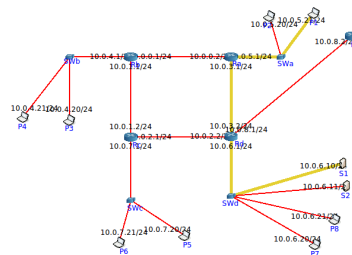


Figura 10. P1 com S1

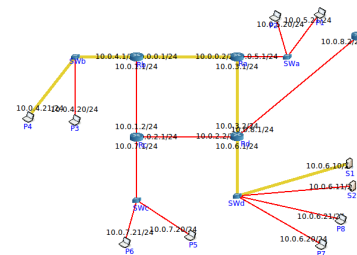


Figura 11. P4 com S1

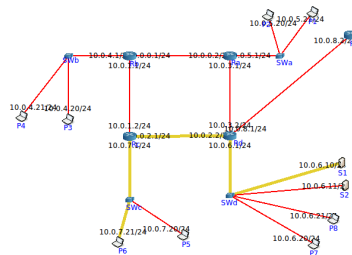


Figura 12. P6 com S1

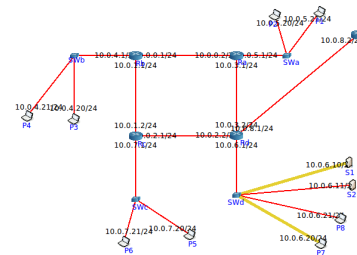


Figura 13. P7 com S1



e)

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

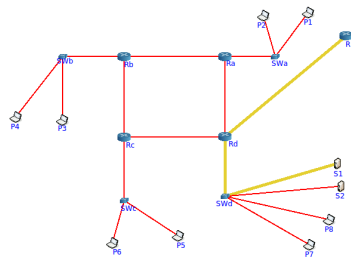


Figura 14. S1 com R

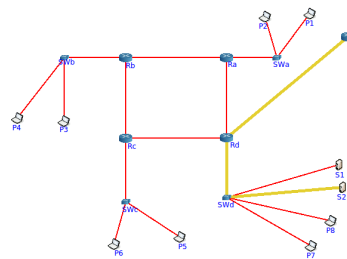


Figura 15. S2 com R

## 2

a)

Para o router:

<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 portáteis da rede 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>

Isto são **tabelas de routing** que ajudam cada dispositivo a descobrir qual 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 gateway 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 gateway é 0.0.0.0, que significa que o

dispositivo 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 gateway será 10.0.4.1, ou seja, o router B reencaminhará o pacote.

**b)**

```
//TODO perguntar isto porque não tenho a certeza (04-11-2017 17:56)//
```

Para verificar se está a ser utilizado encaminhamento estático ou dinâmico, passemos por enviar pacotes entre dispositivos e prestar atenção a possíveis alterações das tabelas de routing dos dispositivos envolvidos.

**c)**

Na shell do servidor 1 executar:

```
route del -net 0.0.0.0
```

Que efetivamente retira a entrada da tabela para destinos que não correspondam aos destinos já existentes. As implicações são as seguintes: O Servidor pode comunicar com quaisquer dispositivos que estejam na sua rede mas não consegue comunicar com quaisquer outros pois o gateway 10.0.6.1 (router D) foi removido. Estas possíveis implicações foram confirmadas com uso de traceroute de dispositivos dentro e fora da rede para o servidor 1.

**d)**

Caso queiramos, inicialmente, restaurar a ligação entre o servidor 1 e a rede B, executar na shell do servidor 1:

```
route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.6.1
```

O router agora saberá reencaminhar pacotes com destino da rede 10.0.4.0 para o router D, reestabelecendo a ligação. Confirma-se que, todavia, ainda existe ligação perdida para as restantes redes da topologia. Sem adicionar um destino 0.0.0.0 (entrada que é usada caso mais nenhuma corresponda), é necessário reestabelecer todas as ligações manualmente, ou seja, executar o comando anterior para as redes 10.0.5.0 e 10.0.7.0 para garantir comunicação entre o servidor e os vários portáteis, e o mesmo para todas as redes dos vários routers.

**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	UG	0	0	0	eth0
10.0.1.0	10.0.6.1	255.255.255.0	UG	0	0	0	eth0
10.0.2.0	10.0.6.1	255.255.255.0	UG	0	0	0	eth0
10.0.3.0	10.0.6.1	255.255.255.0	UG	0	0	0	eth0
10.0.4.0	10.0.6.1	255.255.255.0	UG	0	0	0	eth0
10.0.5.0	10.0.6.1	255.255.255.0	UG	0	0	0	eth0
10.0.6.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
10.0.7.0	10.0.6.1	255.255.255.0	UG	0	0	0	eth0
10.0.8.0	10.0.6.1	255.255.255.0	UG	0	0	0	eth0

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

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

### 3

#### 1)

Começemos por analisar o endereço IP que nos é dado:

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

São dados 8 bits para identificar a rede, e 8 bits para identificar os dispositivos na rede. Podemos manipular estes últimos para criar subredes segundo as necessidades da topologia.

Neste problema, os endereços entre os routers permanece inalterado, pelo que as sub-redes irão-se referir às redes dos departamentos A,B,C e D. Precisamos de garantir, pelo menos, dois hosts utilizáveis para três das redes e quatro hosts utilizáveis para uma das redes. Iremos dividir o nosso endereço em quatro sub endereços com igual capacidade de endereçamento de hosts, pelo que vamos ter de garantir quatro hosts para todas elas, tendo em consideração os dois endereços especiais:

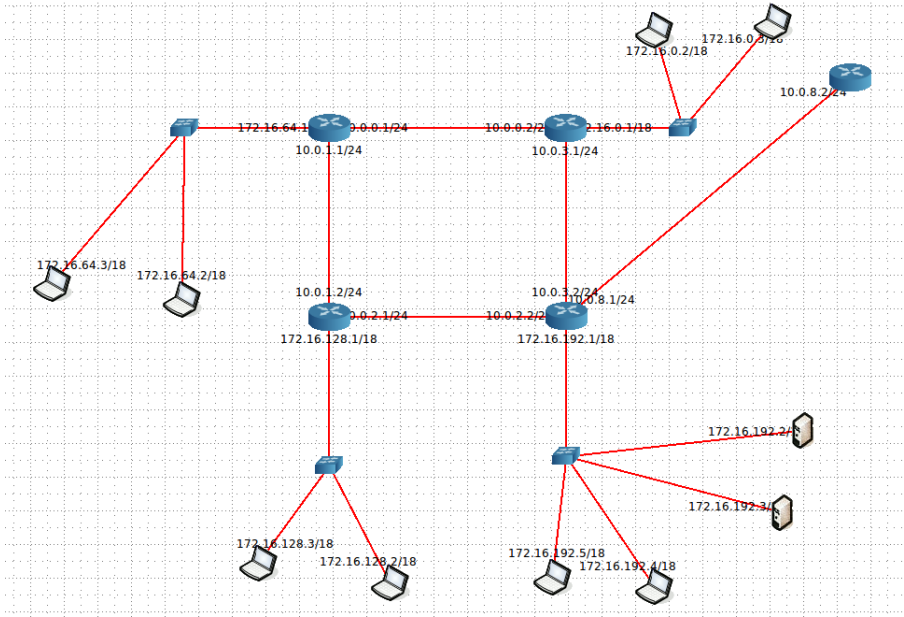
$$\begin{aligned}
 2^n &\geq 4 + 2 && \Leftrightarrow \\
 2^n &\geq 6 \\
 n &\geq \log_2 6 \\
 n &\geq 1.5849625
 \end{aligned}$$

Precisamos de pelo menos dois bits para identificar o host em cada rede. Neste caso iremos dividir a nossa rede em quatro subredes iguais, pelo que precisamos de  $\log_2 4 = 2$  bits para identificar a rede. Assim, "emprestamos" dois bits da parte de endereçamento de host para identificar sub redes. Ficamos com os seguintes endereços:

- Rede A: 172.16.0.0/18
- Rede B: 172.16.64.0/18
- Rede C: 172.16.128.0/18
- Rede D: 172.16.192.0/18

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

A topologia final é a seguinte:



**Figura 16.** Topologia utilizando subnetting

**2)**

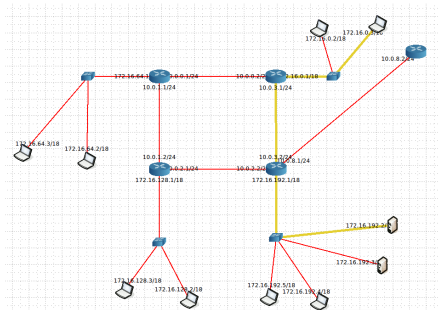
Cada uma das sub redes utiliza 18 bits para identificar a rede, assim:

$$\begin{array}{l} \text{Mascara : } 11111111 \cdot 11111111 \cdot 11|000000 \cdot 00000000 \\ \text{Decimal : } (255) \quad (255) \quad (192) \quad (0) \end{array}$$

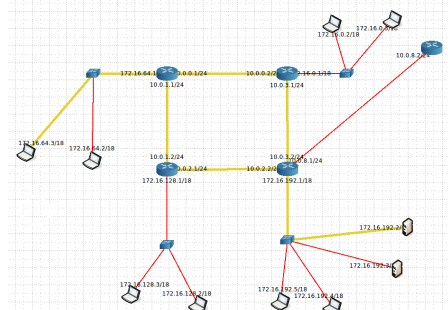
Cada departamento pode endereçar  $2^{14} - 2 = 16382$  dispositivos na sua rede.

**3)**

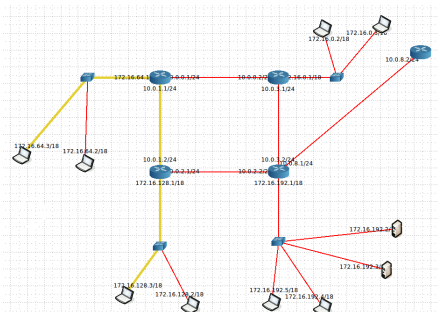
Utilizemos a mesma ferramenta que o CORE disponibiliza para garantir conectividade entre e dentro de departamentos:



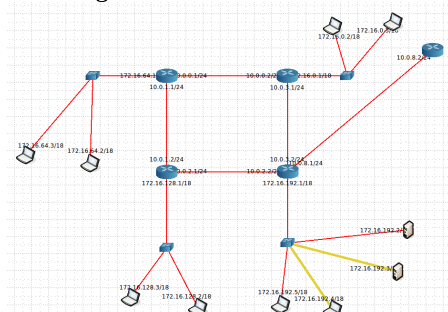
**Figura 17.** P1 com S1



**Figura 18. P4 com S1**



**Figura 19.** P4 com P6



**Figura 20.** P8 com S2