

Redes de Computadores

Trabalho Prático 2: Protocolo IPv4

Ana Rita Peixoto, Sara Queirós, and Sofia Santos

University of Minho, Department of Informatics, 4710-057 Braga, Portugal
e-mail: {a89612,a89491,a89615}@alunos.uminho.pt

Parte I

Exercício 1. Prepare uma topologia CORE para verificar o comportamento do traceroute. Ligue um host (pc) Cliente1 a um router R2; o router R2 a um router R3, que por sua vez, se liga a um host (servidor) Servidor1. (Note que pode não existir conectividade IP imediata entre o Cliente1 e o Servidor1 até que o anúncio de rotas estabilize). Ajuste o nome dos equipamentos atribuídos por defeito para a topologia do enunciado.

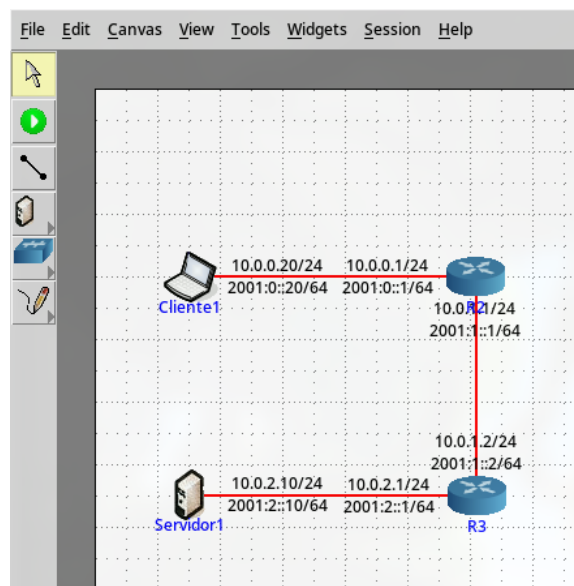


Fig. 1. Topologia CORE.

Alínea a) Ative o wireshark ou o tcpdump no Cliente1. Numa shell do Cliente1, execute o comando `traceroute -I` para o endereço IP do Servidor1.

```
[root@Cliente1 Cliente1.conf]# traceroute -I 10.0.2.10
traceroute to 10.0.2.10 (10.0.2.10), 30 hops max, 60 byte packets
 1 _gateway (10.0.0.1)  0.047 ms  0.005 ms  0.004 ms
 2 10.0.1.2 (10.0.1.2)  0.023 ms  0.006 ms  0.006 ms
 3 10.0.2.10 (10.0.2.10)  0.023 ms  0.008 ms  0.008 ms
```

Fig. 2. Output do comando `traceroute -I 10.0.2.10`

Alínea b) Registe e analise o tráfego ICMP enviado pelo Cliente1 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

O Cliente1 envia, através do protocolo ICMP, 3 packets com um valor crescente de TTL (time-to-live), começando em 1, até obter uma resposta do Servidor1.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	fe80::f4ee:4c:ff:f...	ff02::2	ICMPv6	70	Router Solicitation from fe8e:4c:04:62:9d
2	0.210004769	fe80::200:ff:feaa...	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:aa:00:00
3	0.423338298	fe80::7caf:1b:ff:f...	ff02::2	ICMPv6	70	Router Solicitation from 7e:ef:8b:1b:5a:c2
4	1.302189877	10.0.0.1	224.0.0.5	OSPF	70	Hello Packet
5	3.302378384	10.0.0.1	224.0.0.5	OSPF	70	Hello Packet
6	5.302453914	10.0.0.1	224.0.0.5	OSPF	70	Hello Packet
7	5.308615826	fe80::200:ff:feaa...	ff02::5	OSPF	90	Hello Packet
8	7.250617087	fe80::f4ee:4c:ff:f...	ff02::2	ICMPv6	70	Router Solicitation from fe8e:4c:04:62:9d
9	7.302516439	10.0.0.1	224.0.0.5	OSPF	70	Hello Packet
10	8.530016745	fe80::200:ff:feaa...	ff02::2	ICMPv6	70	Router Solicitation from 00:00:00:aa:00:00
11	9.991878338	00:00:00:aa:00:00	Broadcast	ARP	42	Who has 10.0.0.1? Tell 10.0.0.20
12	9.991210198	00:00:00:aa:00:01	00:00:00:aa:00:00	ARP	42	10.0.0.1 is at 00:00:00:aa:00:01
13	8.991212078	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=2/256, ttl=1 (no response found!)
14	8.991222222	10.0.0.2	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=2/256, ttl=1 (no response found!)
15	8.991222288	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=2/256, ttl=1 (no response found!)
16	8.991235568	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=3/768, ttl=1 (no response found!)
17	8.991235568	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=3/768, ttl=1 (no response found!)
18	8.991241508	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=4/1024, ttl=2 (no response found!)
19	8.991241508	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=4/1024, ttl=2 (no response found!)
20	8.991241508	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=4/1024, ttl=2 (no response found!)
21	8.991241508	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=4/1024, ttl=2 (no response found!)
22	8.991241508	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=4/1024, ttl=2 (no response found!)
23	8.991241508	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=4/1024, ttl=2 (no response found!)
24	8.991241508	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=4/1024, ttl=2 (no response found!)
25	8.991281758	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=7/1792, ttl=3 (reply in 26)
26	8.991303748	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=7/1792, ttl=3 (reply in 26)
27	8.991303748	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=8/2048, ttl=3 (reply in 28)
28	8.991313478	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=8/2048, ttl=3 (reply in 27)
29	8.991316068	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=9/2304, ttl=3 (reply in 30)
30	8.991322158	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=9/2304, ttl=3 (reply in 29)
31	8.991325178	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=10/2560, ttl=4 (reply in 32)
32	8.991331278	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=10/2560, ttl=4 (reply in 31)
33	8.991333968	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=11/2816, ttl=4 (reply in 33)
34	8.991339818	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=11/2816, ttl=4 (reply in 33)
35	8.991342348	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=12/3072, ttl=4 (reply in 36)
36	8.991348218	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=12/3072, ttl=4 (reply in 35)
37	8.991350968	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=13/3328, ttl=5 (reply in 37)
38	8.991356878	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=13/3328, ttl=5 (reply in 37)
39	8.991359368	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=14/3584, ttl=5 (reply in 40)
40	8.991365248	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=14/3584, ttl=5 (reply in 39)
41	8.991367988	10.0.0.20	10.0.0.2.10	ICMP	74	Echo (ping) request id=8x0015, seq=15/3840, ttl=5 (reply in 42)

Fig. 3. Tráfego ICMP enviado e recebido pelo Cliente1.

Alínea c) Qual deve ser o valor inicial mínimo do campo TTL para alcançar o Servidor1? Verifique na prática que a sua resposta está correta.

Para alcançar o Servidor1, o valor mínimo de TTL deve ser 3. O tráfego ICMP da alínea anterior mostra-nos precisamente isso, o Cliente1 apenas recebe uma resposta do Servidor1 quando o valor de TTL chega a 3.

Alínea d) Calcule o valor médio do tempo de ida-e-volta (Round-Trip Time) obtido.

Após efetuar várias medições, obtivemos um Round-Trip Time médio de 0,047 ms. Porém, verificamos várias oscilações neste tempo, com valores entre 0,008 ms e 0,169 ms.

Exercício 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 (expanda o tab correspondente na janela de detalhe do wireshark). Através da análise do cabeçalho IP diga:

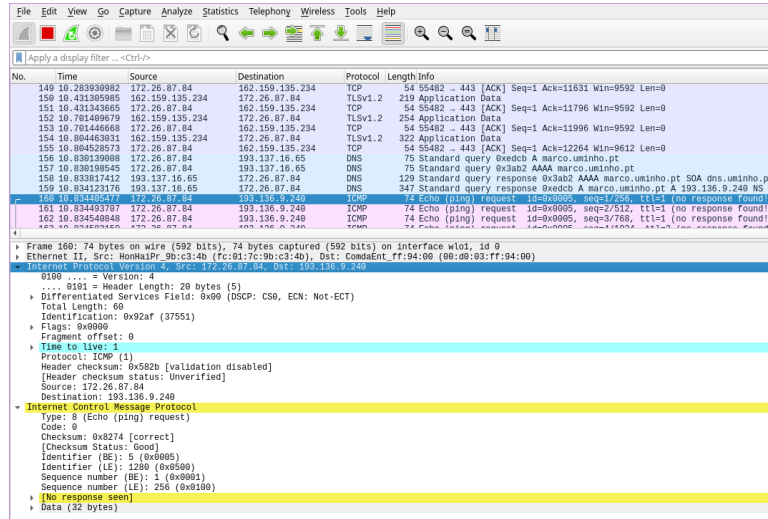


Fig. 4. Tab do wireshark correspondente à primeira mensagem ICMP capturada.

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

172.26.87.84

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

O campo identifica o protocolo usado para enviar a mensagem, que neste caso é ICMP (1).

Alínea 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?

O cabeçalho tem 20 bytes. Como o tamanho total é 60 bytes, o tamanho do payload deve ser $60 - 20 = 40$ bytes.

Alínea d) O datagrama IP foi fragmentado? Justifique.

Podemos ver que o campo "Fragment offset" tem o valor 0, logo ou este datagrama não foi fragmentado ou este é o primeiro fragmento. Porém, como a flag "More fragments" tem valor 0, podemos concluir que, não havendo mais fragmentos, o datagrama não foi fragmentado.

```

Flags: 0x0000
 0... .. = Reserved bit: Not set
.0... .. = Don't fragment: Not set
..0... .. = More fragments: Not set

```

Fig. 5. Campo "flags" referente à tab do wireshark anterior.

Alínea 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.

Time	Source	Destination	Protocol	Length	Info
182.10.836676299	172.26.254.254	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
180.10.834405477	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=1/256, ttl=1 (no response found!)
161.10.834493707	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=2/512, ttl=1 (no response found!)
162.10.834540848	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=3/768, ttl=1 (no response found!)
163.10.834583150	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=4/1024, ttl=2 (no response found!)
164.10.834617192	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=5/1280, ttl=2 (no response found!)
165.10.834647754	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=6/1536, ttl=2 (no response found!)
166.10.834680955	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=7/1792, ttl=3 (no response found!)
167.10.834710345	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=8/2048, ttl=3 (no response found!)
168.10.834740254	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=9/2304, ttl=3 (no response found!)
169.10.834773818	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=10/2560, ttl=4 (reply in 183)
170.10.834804817	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=11/2816, ttl=4 (reply in 185)
171.10.834833950	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=12/3072, ttl=4 (reply in 186)
172.10.834867919	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=13/3328, ttl=5 (reply in 187)
173.10.834897199	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=14/3584, ttl=5 (reply in 188)
174.10.834927199	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=15/3840, ttl=5 (reply in 189)
175.10.834960100	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=16/4096, ttl=6 (reply in 190)
177.10.835827688	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=17/4352, ttl=6 (reply in 197)
191.10.837061788	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=18/4608, ttl=6 (reply in 198)
194.10.837208470	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=19/4864, ttl=7 (reply in 199)
195.10.837273517	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=20/5120, ttl=7 (reply in 200)
196.10.837317062	172.26.87.84	193.136.9.240	ICMP	74	Echo (ping) request id=0x0005, seq=21/5376, ttl=7 (reply in 201)
183.10.836676526	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=10/2560, ttl=61 (request in 169)

Fig. 6. Tráfego ICMP enviado pelo nosso computador.

Os campos "Identification", "Time to live" e "Header checksum".

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

O valor da Identificação incrementa 1 por cada nova mensagem e o primeiro byte é sempre igual - "0x92". No campo TTL verifica-se incrementos de 1 a cada 3 mensagens.

Alínea 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?

Time	Source	Destination	Protocol	Length	Info
10.835741014	172.16.2.1	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.835953550	172.16.2.1	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.835983755	172.26.254.254	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.835953995	172.16.2.1	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.835954066	172.26.254.254	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.836010229	172.26.254.254	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.836015526	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=10/2560, ttl=61 (request in 169)
10.836016631	172.16.115.252	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.836793277	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=11/2816, ttl=61 (request in 170)
10.836793410	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=12/3072, ttl=61 (request in 171)
10.836793522	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=13/3328, ttl=61 (request in 172)
10.836793709	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=14/3584, ttl=61 (request in 173)
10.836793852	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=15/3840, ttl=61 (request in 174)
10.836794007	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=16/4096, ttl=61 (request in 175)
10.837157602	172.16.115.252	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.837157840	172.16.115.252	172.26.87.84	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
10.838536836	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=17/4352, ttl=61 (request in 177)
10.838867354	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=18/4608, ttl=61 (request in 191)
10.838867578	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=19/4864, ttl=61 (request in 194)
10.838867677	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=20/5120, ttl=61 (request in 195)
10.838867771	193.136.9.240	172.26.87.84	ICMP	74	Echo (ping) reply id=0x0005, seq=21/5376, ttl=61 (request in 196)

Fig. 7. Tráfego ICMP recebido pelo nosso computador. As mensagens TTL exceeded são as que têm fundo preto.

O valor do campo TTL Exceeded varia entre 253 e 255 uma vez que ele passa por 3 routers intermédios então decremente 3 vezes.

Exercício 3. Pretende-se agora analisar a fragmentação de pacotes IP. Reponha a ordem do tráfego capturado usando a coluna do tempo de captura. Observe o tráfego depois do tamanho de pacote ter sido definido para 3256 bytes.

A primeira mensagem ICMP foi fragmentada porque o seu tamanho era demasiado grande para ser transmitida de uma só vez. Só se pode transmitir 1500 bytes de uma vez e a mensagem possuía 3236 bytes.

10.000000	172.26.53.225	172.26.254.254	NBNS	92 Name query NBSTAT *	0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00
2.124663	172.26.53.225	172.16.115.252	NBNS	92 Name query NBSTAT *	0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00
3.124849	172.26.53.225	172.16.2.1	NBNS	92 Name query NBSTAT *	0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00-0x00
4.854248	172.159.130.234		TLSv1.2	194 Application Data	
5.854248	172.159.130.234		TLSv1.2	193 Application Data	
6.854554	172.26.53.225	172.159.130.234	TCP	5454787 > 443 [ACK] Seq=100 Ack=150 Win=150 Len=0	
7.912039	172.26.53.225	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=TCP, l, off=0, Dst=4899) [Reassembled in #9]	
8.912039	172.26.53.225	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=TCP, l, off=1480, Dst=4899) [Reassembled in #9]	
9.912039	172.26.53.225	193.136.9.240	TCP	310 Echo (ping) request id=0x0001, seq=27749/25664, ttl=255 (reply in 18)	
10.961920	172.26.53.225	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=TCP, l, off=0, Dst=4898) [Reassembled in #12]	
11.961920	172.26.53.225	193.136.9.240	IPv4	1514 Fragmented IP protocol (proto=TCP, l, off=0, Dst=4898) [Reassembled in #12]	
12.961920	172.26.53.225	193.136.9.240	TCP	210 Echo (ping) request id=0x0001, seq=27750/26220, ttl=64 (no response found)	
13.962493	193.136.9.240	172.26.53.225	IPv4	1514 Fragmented IP protocol (proto=TCP, l, off=0, Dst=2940) [Reassembled in #18]	
14.985115	193.136.9.240	172.26.53.225	IPv4	310 Fragmented IP protocol (proto=TCP, l, off=7360, Dst=7349) [Reassembled in #18]	

Fig. 8. Fragmentação do pacote inicial

```

  ▾ Internet Protocol Version 4, Src: 172.26.53.225, 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: 1500
    Identification: 0x4899 (18585)
  ▾ Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
    Fragment Offset: 0
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0xa013 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.53.225
    Destination Address: 193.136.9.240
    \[Reassembled IPv4 in frame: 9\]
  > Data (1480 bytes)

```

Fig. 9. Primeiro Fragmento do Datagrama IP Segmentado

Como o campo “fragment offset” tem o valor zero, podemos concluir que se trata do primeiro fragmento. O tamanho deste datagrama IP é 1500 bytes.

Alínea 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?

O identificador deste fragmento e do anterior é o mesmo, logo podemos concluir que ambos são fragmentos do mesmo datagrama. A partir da observação do campo “fragment offset”, podemos concluir que não se trata do 1º fragmento, dado que esse valor não é zero. Há mais fragmentos porque o bit da flag “more fragments” está a 1.

```

v Internet Protocol Version 4, Src: 172.26.53.225, 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: 1500
    Identification: 0x4899 (18585)
  v Flags: 0x20, More fragments
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..1. .... = More fragments: Set
    Fragment Offset: 1480
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0x9f5a [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.53.225
    Destination Address: 193.136.9.240
    [Reassembled IPv4 in frame: 9]
  > Data (1480 bytes)

```

Fig. 10. Segundo fragmento do datagrama IP segmentado.

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

Foram criados 3 fragmentos a partir do datagrama original, pois apenas 2 outros fragmentos possuem um valor no campo "Identification" igual ao primeiro fragmento. Tal como podemos ver na seguinte figura, a flag “more fragments” do 3º fragmento tem o seu bit a 0, logo este fragmento deve ser o último.

```

v Internet Protocol Version 4, Src: 172.26.53.225, 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: 296
    Identification: 0x4899 (18585)
  v Flags: 0x01
    0... .... = Reserved bit: Not set
    .0.. .... = Don't fragment: Not set
    ..0. .... = More fragments: Not set
    Fragment Offset: 2960
    Time to Live: 255
    Protocol: ICMP (1)
    Header Checksum: 0xc355 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.26.53.225
    Destination Address: 193.136.9.240
  > [3 IPv4 Fragments (3236 bytes): #7(1480), #8(1480), #9(276)]
  > Internet Control Message Protocol

```

Fig. 11. Equipamentos existentes

Alínea 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.

Os campos que mudam no cabeçalho IP entre os diferentes fragmentos são: header checksum, fragment offset e total length. Essa informação permite reconstruir o datagrama original porque todos os fragmentos possuem o mesmo identificador (permite concluir que pertencem ao mesmo datagrama). Os campos fragment offset e total length informam-nos onde começa cada fragmento e o tamanho que cada um ocupa.

Parte II

Exercício 1. Atenda aos endereços IP atribuídos automaticamente pelo CORE aos diversos equipamentos da topologia.

Alínea 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.

Os endereços IP atribuídos a cada equipamento estão descritos na imagem abaixo. A máscara de rede utilizada foi de 24 bits.

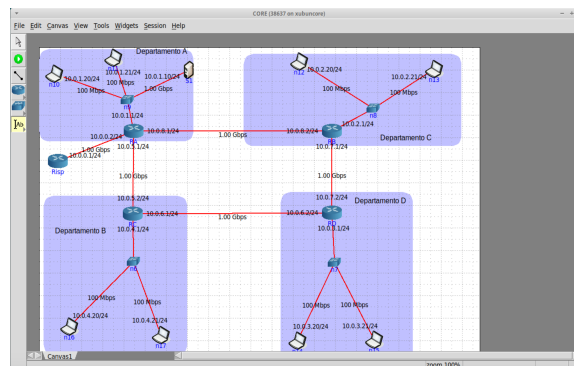


Fig. 12. Topologia core com os respectivos endereços IP

Alínea b) Tratam-se de endereços públicos ou privados? Porquê?

Tratam-se de endereços privados pois encontram-se no range de endereços privados.

Alínea c) Porque razão não é atribuído um endereço IP aos switches?

Um switch não funciona ao nível de rede, mas sim ao nível de ligação de dados. Assim, funciona como uma ponte e conecta todos os elementos da rede. Apenas opera sobre endereços físicos e um endereço IP é um endereço lógico.

Alínea 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).

Após efetuar o comando ping para 1 computador de cada departamento, verificou-se existência de conectividade entre os dois dispositivos, tal como é visível na imagem abaixo.

```

root@S1:/tmp/pycore.38637/S1.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_seq=1 ttl=62 time=0.117 ms
64 bytes from 10.0.4.20: icmp_seq=2 ttl=62 time=0.077 ms
64 bytes from 10.0.4.20: icmp_seq=3 ttl=62 time=0.122 ms
64 bytes from 10.0.4.20: icmp_seq=4 ttl=62 time=0.126 ms
^C
--- 10.0.4.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 307ms
rtt min/avg/max/mdev = 0.077/0.110/0.126/0.022 ms
root@S1:/tmp/pycore.38637/S1.conf# ping 10.0.2.21
PING 10.0.2.21 (10.0.2.21) 56(84) bytes of data:
64 bytes from 10.0.2.21: icmp_seq=1 ttl=62 time=0.181 ms
64 bytes from 10.0.2.21: icmp_seq=2 ttl=62 time=0.095 ms
64 bytes from 10.0.2.21: icmp_seq=3 ttl=62 time=0.218 ms
64 bytes from 10.0.2.21: icmp_seq=4 ttl=62 time=0.100 ms
64 bytes from 10.0.2.21: icmp_seq=5 ttl=62 time=0.086 ms
^C
--- 10.0.2.21 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 410ms
rtt min/avg/max/mdev = 0.086/0.136/0.218/0.053 ms
root@S1:/tmp/pycore.38637/S1.conf# ping 10.0.3.20
PING 10.0.3.20 (10.0.3.20) 56(84) bytes of data:
64 bytes from 10.0.3.20: icmp_seq=1 ttl=61 time=0.113 ms
64 bytes from 10.0.3.20: icmp_seq=2 ttl=61 time=0.385 ms
64 bytes from 10.0.3.20: icmp_seq=3 ttl=61 time=0.094 ms
64 bytes from 10.0.3.20: icmp_seq=4 ttl=61 time=0.094 ms
64 bytes from 10.0.3.20: icmp_seq=5 ttl=61 time=0.108 ms
64 bytes from 10.0.3.20: icmp_seq=6 ttl=61 time=0.104 ms
^C
--- 10.0.3.20 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 512ms
rtt min/avg/max/mdev = 0.094/0.149/0.385/0.106 ms
root@S1:/tmp/pycore.38637/S1.conf# ping 10.0.1.20
PING 10.0.1.20 (10.0.1.20) 56(84) bytes of data:
64 bytes from 10.0.1.20: icmp_seq=1 ttl=64 time=0.041 ms
64 bytes from 10.0.1.20: icmp_seq=2 ttl=64 time=0.078 ms
64 bytes from 10.0.1.20: icmp_seq=3 ttl=64 time=0.063 ms
64 bytes from 10.0.1.20: icmp_seq=4 ttl=64 time=0.065 ms
64 bytes from 10.0.1.20: icmp_seq=5 ttl=64 time=0.063 ms
^C
--- 10.0.1.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 410ms
rtt min/avg/max/mdev = 0.041/0.062/0.078/0.011 ms
root@S1:/tmp/pycore.38637/S1.conf# ]

```

Fig. 13. Verificação de existência de conectividade entre o departamento B e o servidor

Alínea e) Verifique se existe conectividade IP do router de acesso RISP para o servidor S1.

Analogamente à alínea anterior, através do comando ping conseguimos testar a conectividade entre o router ISP e o servidor do departamento A. Tal como é visível pela imagem, existe conectividade entre ambos.


```

root@Risp:/tmp/pycore.38637/Risp.conf# ping 10.0.1.10
PING 10.0.1.10 (10.0.1.10) 56(84) bytes of data:
64 bytes from 10.0.1.10: icmp_seq=1 ttl=63 time=0.063 ms
64 bytes from 10.0.1.10: icmp_seq=2 ttl=63 time=0.105 ms
64 bytes from 10.0.1.10: icmp_seq=3 ttl=63 time=0.100 ms
64 bytes from 10.0.1.10: icmp_seq=4 ttl=63 time=0.082 ms
64 bytes from 10.0.1.10: icmp_seq=5 ttl=63 time=0.105 ms
64 bytes from 10.0.1.10: icmp_seq=6 ttl=63 time=0.070 ms
64 bytes from 10.0.1.10: icmp_seq=7 ttl=63 time=0.083 ms
64 bytes from 10.0.1.10: icmp_seq=8 ttl=63 time=0.105 ms
64 bytes from 10.0.1.10: icmp_seq=9 ttl=63 time=0.079 ms
64 bytes from 10.0.1.10: icmp_seq=10 ttl=63 time=0.104 ms
^C
--- 10.0.1.10 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9207ms
rtt min/avg/max/mdev = 0.063/0.089/0.105/0.018 ms
root@Risp:/tmp/pycore.38637/Risp.conf#

```

Fig. 14. Verificação de existência de conectividade entre o router ISP e o servidor

Exercício 2. Para o router e um laptop do departamento C:

Alínea 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`).

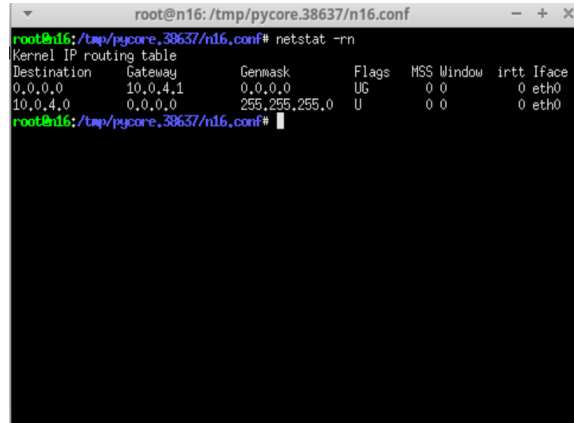
O campo `destination` da tabela de encaminhamento refere-se aos endereços IP destino. Assim, para cada endereço IP destino presente na tabela, existe uma porta para saída do router (gateway). A coluna `Genmask` refere-se à máscara utilizada. Neste caso, a máscara `255.255.255.0` equivale a utilizar 24 bits como máscara de rede. As flags permitem identificar se se trata de uma rede local ou global. A última coluna presente na tabela refere qual a interface de saída da máquina local.

```

root@RC:/tmp/pycore.38637/RC.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.0.0 10.0.5.1 255.255.255.0 UG 0 0 0 eth0
10.0.1.0 10.0.5.1 255.255.255.0 UG 0 0 0 eth0
10.0.2.0 10.0.5.1 255.255.255.0 UG 0 0 0 eth0
10.0.3.0 10.0.6.2 255.255.255.0 UG 0 0 0 eth2
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth1
10.0.5.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.6.0 0.0.0.0 255.255.255.0 U 0 0 0 eth2
10.0.7.0 10.0.6.2 255.255.255.0 UG 0 0 0 eth2
10.0.8.0 10.0.5.1 255.255.255.0 UG 0 0 0 eth0
root@RC:/tmp/pycore.38637/RC.conf#

```

Fig. 15. Tabela de encaminhamento do router C



```

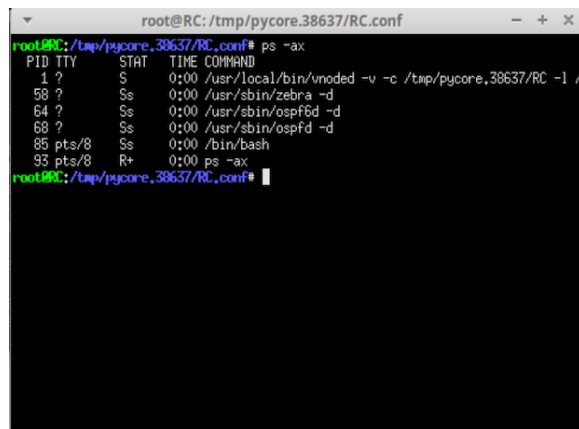
root@n16: /tmp/pycore.38637/n16.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.4.1 0.0.0.0 UG 0 0 0 eth0
10.0.4.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@n16: /tmp/pycore.38637/n16.conf#

```

Fig. 16. Tabela de encaminhamento de um laptop da rede C

Alínea b) Diga, justificando, se está a ser usado encaminhamento estático ou dinâmico (sugestão: analise que processos estão a correr em cada sistema, por exemplo, ps -ax).

Está a ser usado encaminhamento dinâmico. Através da imagem conseguimos observar que está a ser executado o protocolo OSPF6, que se trata de um protocolo de encaminhamento dinâmico.



```

root@RC: /tmp/pycore.38637/RC.conf# ps -ax
PID TTY STAT TIME COMMAND
1 ? S 0:00 /usr/local/bin/vnoded -v -c /tmp/pycore.38637/RC -1 /
58 ? Ss 0:00 /usr/sbin/zebra -d
64 ? Ss 0:00 /usr/sbin/ospf6d -d
68 ? Ss 0:00 /usr/sbin/ospf6d -d
85 pts/8 Ss 0:00 /bin/bash
93 pts/8 R+ 0:00 ps -ax
root@RC: /tmp/pycore.38637/RC.conf#

```

Fig. 17. Tabela de encaminhamento de um laptop da rede C

Alínea 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 implicações tem esta medida para os utilizadores da organização MIEI-RC que acedem ao servidor. Justifique.

Ao eliminar definitivamente a rota por defeito, o servidor 1 apenas saberá encaminhar tráfego de pacotes cujo endereço IP destino é 10.0.1.0, ou seja, para a sua rede local. Não é possível efetuar comunicação em subredes com outro endereço IP. Ao contrário do que se verificou anteriormente, que existia comunicação entre o servidor da subrede A e um laptop de qualquer subrede da rede, agora não existe essa comunicação, perdendo toda a conectividade, tal como é visível na figura 18.

```

root@S1: /tmp/pycore.38637/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
0.0.0.0 10.0.1.1 0.0.0.0 UG 0 0 0 eth0
10.0.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@S1: /tmp/pycore.38637/S1.conf# route delete default
root@S1: /tmp/pycore.38637/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@S1: /tmp/pycore.38637/S1.conf#

```

Fig. 18. Eliminação da rota por defeito no servidor 1

```

root@S1: /tmp/pycore.38637/S1.conf# route delete default
root@S1: /tmp/pycore.38637/S1.conf# ping 10.0.2.20
connect: Network is unreachable
root@S1: /tmp/pycore.38637/S1.conf#

```

Fig. 19. Teste de ping após eliminação da rota por defeito

Alínea 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.

```

root@S1: /tmp/pycore.38637/S1.conf# route
Kernel IP routing table
Destination Gateway Genmask Flags Metric Ref Use Iface
10.0.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
root@S1: /tmp/pycore.38637/S1.conf# route add 10.0.1.10 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.conf# route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.conf#

```

Fig. 20. Adição das rotas estáticas

No servidor S1:

```

route add 10.0.1.0 gw 10.0.1.1
route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.1.1
route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.1.1
route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.1.1

```

Alínea e) 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.

```

root@S1: /tmp/pycore.38637/S1.conf
root@S1: /tmp/pycore.38637/S1.conf# route add 10.0.1.10 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.conf# route add -net 10.0.4.0 netmask 255.255.255.0 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.conf# route add -net 10.0.3.0 netmask 255.255.255.0 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.conf# route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.1.1
root@S1: /tmp/pycore.38637/S1.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_seq=1 ttl=62 time=0.149 ms
64 bytes from 10.0.4.20: icmp_seq=2 ttl=62 time=0.123 ms
64 bytes from 10.0.4.20: icmp_seq=3 ttl=62 time=0.104 ms
64 bytes from 10.0.4.20: icmp_seq=4 ttl=62 time=0.124 ms
^C
--- 10.0.4.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3073ms
rtt min/avg/max/mdev = 0.104/0.125/0.149/0.016 ms
root@S1: /tmp/pycore.38637/S1.conf# ping 10.0.3.20
PING 10.0.3.20 (10.0.3.20) 56(84) bytes of data.
64 bytes from 10.0.3.20: icmp_seq=1 ttl=61 time=0.125 ms
64 bytes from 10.0.3.20: icmp_seq=2 ttl=61 time=0.139 ms
64 bytes from 10.0.3.20: icmp_seq=3 ttl=61 time=0.104 ms
64 bytes from 10.0.3.20: icmp_seq=4 ttl=61 time=0.155 ms
64 bytes from 10.0.3.20: icmp_seq=5 ttl=61 time=0.123 ms
^C
--- 10.0.3.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4112ms
rtt min/avg/max/mdev = 0.104/0.129/0.155/0.018 ms
root@S1: /tmp/pycore.38637/S1.conf# ping 10.0.2.20
PING 10.0.2.20 (10.0.2.20) 56(84) bytes of data.
64 bytes from 10.0.2.20: icmp_seq=1 ttl=62 time=0.119 ms
64 bytes from 10.0.2.20: icmp_seq=2 ttl=62 time=0.123 ms
64 bytes from 10.0.2.20: icmp_seq=3 ttl=62 time=0.395 ms
64 bytes from 10.0.2.20: icmp_seq=4 ttl=62 time=0.094 ms
64 bytes from 10.0.2.20: icmp_seq=5 ttl=62 time=0.092 ms
^C
--- 10.0.2.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4190ms
rtt min/avg/max/mdev = 0.092/0.164/0.395/0.116 ms
root@S1: /tmp/pycore.38637/S1.conf# ping 10.0.1.20
PING 10.0.1.20 (10.0.1.20) 56(84) bytes of data.
64 bytes from 10.0.1.20: icmp_seq=1 ttl=64 time=0.085 ms
64 bytes from 10.0.1.20: icmp_seq=2 ttl=64 time=0.065 ms
64 bytes from 10.0.1.20: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 10.0.1.20: icmp_seq=4 ttl=64 time=0.075 ms
^C
--- 10.0.1.20 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3076ms
rtt min/avg/max/mdev = 0.060/0.071/0.085/0.011 ms
root@S1: /tmp/pycore.38637/S1.conf#

```

Fig. 21. Teste de ping entre o S1 e um computador de cada departamento

```

root@S1: /tmp/pycore.38637/S1.conf
root@S1: /tmp/pycore.38637/S1.conf# netstat -rn
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
10.0.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
10.0.1.10 10.0.1.1 255.255.255.255 UGH 0 0 0 eth0
10.0.2.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
10.0.3.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
10.0.4.0 10.0.1.1 255.255.255.0 UG 0 0 0 eth0
root@S1: /tmp/pycore.38637/S1.conf#

```

Fig. 22. Tabela de encaminhamento de S1

Exercício 3. Considere a topologia definida anteriormente. Assuma que o endereçamento entre os routers se mantém inalterado, contudo, o endereçamento em cada departamento deve ser redefinido.

Alínea a) Considere que dispõe apenas do endereço de rede IP 130.XX.96.0/19, 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. Assuma que todos os endereços de sub-redes são usáveis. Deve justificar as opções usadas.

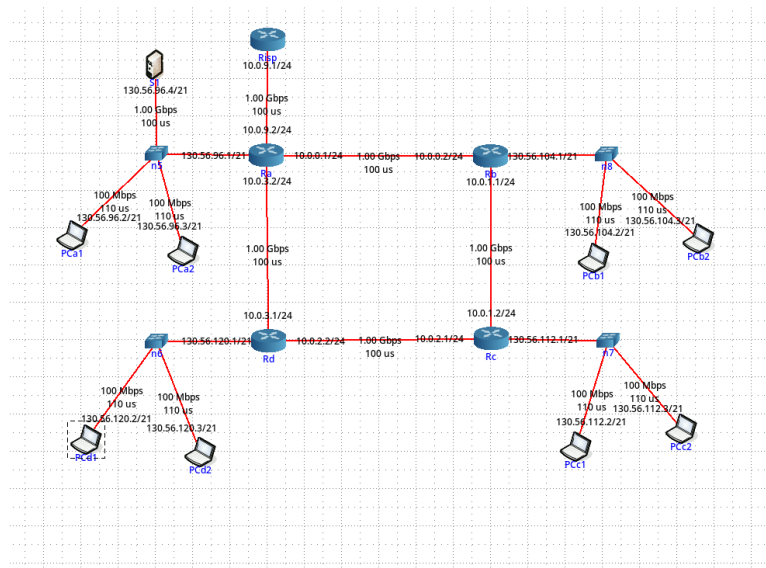


Fig. 23. Topologia com sub-redes.

Como temos 4 sub-redes, precisamos de 2 bits para as distinguir. Assim, ficamos com uma nova máscara de 21 bits, dos quais 2 identificam a sub-rede. A partir daqui apenas é preciso atribuir um endereço IP a cada departamento com base na nova máscara, mudando os bits correspondentes à sub-rede. Na figura acima é possível ver um esquema de endereçamento com 4 sub-redes.

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

Usámos a máscara /21 = 255.255.248.0

Como nos sobram 11 bits para os endereços dos hosts, podemos interligar $2^{11} - 2 = 2048 - 2 = 2046$ hosts em cada departamento (temos que retirar 2 ao valor final devido ao endereço de broadcast e ao endereço de rede).

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

```

[root@PCa1 PCa1.conf]# ping 10.0.9.1
PING 10.0.9.1 (10.0.9.1) 56(84) bytes of data.
64 bytes from 10.0.9.1: icmp_seq=1 ttl=63 time=0.767 ms
64 bytes from 10.0.9.1: icmp_seq=2 ttl=63 time=0.448 ms
64 bytes from 10.0.9.1: icmp_seq=3 ttl=63 time=0.364 ms
64 bytes from 10.0.9.1: icmp_seq=4 ttl=63 time=0.359 ms
^C
--- 10.0.9.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3034ms
rtt min/avg/max/mdev = 0.359/0.484/0.767/0.166 ms
[root@PCa1 PCa1.conf]#

[root@PCb1 PCb1.conf]# ping 10.0.9.1
PING 10.0.9.1 (10.0.9.1) 56(84) bytes of data.
64 bytes from 10.0.9.1: icmp_seq=1 ttl=62 time=1.11 ms
64 bytes from 10.0.9.1: icmp_seq=2 ttl=62 time=0.676 ms
64 bytes from 10.0.9.1: icmp_seq=3 ttl=62 time=0.678 ms
64 bytes from 10.0.9.1: icmp_seq=4 ttl=62 time=0.815 ms
^C
--- 10.0.9.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 0.676/0.820/1.114/0.178 ms
[root@PCb1 PCb1.conf]#

[root@PCc1 PCc1.conf]# ping 10.0.9.1
PING 10.0.9.1 (10.0.9.1) 56(84) bytes of data.
64 bytes from 10.0.9.1: icmp_seq=1 ttl=61 time=1.34 ms
64 bytes from 10.0.9.1: icmp_seq=2 ttl=61 time=0.980 ms
64 bytes from 10.0.9.1: icmp_seq=3 ttl=61 time=1.04 ms
64 bytes from 10.0.9.1: icmp_seq=4 ttl=61 time=1.03 ms
^C
--- 10.0.9.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 0.980/1.095/1.340/0.142 ms
[root@PCc1 PCc1.conf]#

[root@PCd1 PCd1.conf]# ping 10.0.9.1
PING 10.0.9.1 (10.0.9.1) 56(84) bytes of data.
64 bytes from 10.0.9.1: icmp_seq=1 ttl=62 time=1.11 ms
64 bytes from 10.0.9.1: icmp_seq=2 ttl=62 time=0.676 ms
64 bytes from 10.0.9.1: icmp_seq=3 ttl=62 time=0.673 ms
64 bytes from 10.0.9.1: icmp_seq=4 ttl=62 time=0.728 ms
^C
--- 10.0.9.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3019ms
rtt min/avg/max/mdev = 0.673/0.795/1.106/0.180 ms
[root@PCd1 PCd1.conf]#

```

Fig. 24. Comando ping executado por computadores de todos os departamentos ao router do ISP.

Como podemos ver na figura acima, todos os departamentos conseguem ligar-se ao router do ISP, mantendo-se assim a conectividade IP. De modo análogo, a conectividade entre os departamentos também se mantém. Podemos ver na figura abaixo um exemplo de conectividade entre o router do departamento A e o departamento B. O mesmo se verifica para os outros routers e departamentos.

```

[root@Ra Ra.conf]# ping 130.56.104.1
PING 130.56.104.1 (130.56.104.1) 56(84) bytes of data.
64 bytes from 130.56.104.1: icmp_seq=1 ttl=64 time=0.316 ms
64 bytes from 130.56.104.1: icmp_seq=2 ttl=64 time=0.307 ms
64 bytes from 130.56.104.1: icmp_seq=3 ttl=64 time=0.318 ms
64 bytes from 130.56.104.1: icmp_seq=4 ttl=64 time=0.334 ms
^C
--- 130.56.104.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3028ms
rtt min/avg/max/mdev = 0.307/0.318/0.334/0.009 ms
[root@Ra Ra.conf]#

```

Fig. 25. Comando ping executado pelo router do departamento A ao departamento B.

Conclusão

Através deste trabalho prático fomos capazes de consolidar aquilo que aprendemos nas aulas teóricas, nomeadamente datagramas IP, fragmentação, endereçamento, encaminhamento e subnetting. Também aprendemos a usar software de gestão e monitorização de redes, como o CORE e o wireshark, para além de comandos como o traceroute ou o ping. Estes conhecimentos ser-nos-ão bastante valiosos, não só no contexto da UC e do curso, mas também no nosso dia a dia e na nossa vida profissional.