Redes de Computadores – TP4

Protocolo IPv4 (Parte I)

1.

a. Active o wireshark ou o tcpdump no host n4. Numa Shell de n4, execute o comando traceroute -I para o endereço IP do host n1.

```
root@n4:/tmp/pycore.40165/n4.conf# traceroute -I 10.0.0.10
traceroute to 10.0.0.10 (10.0.0.10), 30 hops max, 60 byte packets
 1  10.0.2.1 (10.0.2.1)  0.075 ms  0.006 ms  0.005 ms
 2  10.0.1.1 (10.0.1.1)  0.021 ms  0.007 ms  0.007 ms
 3  10.0.0.10 (10.0.0.10)  0.025 ms  0.010 ms  0.010 ms
```

b. Registe e analise o tráfego ICMP enviado por n4 e o tráfego ICMP recebido como resposta. Comente os resultados face ao comportamento esperado.

```
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:55:44.111697 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:55:44.111966 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:55:54.103694 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:55:54.116620 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:56:04.100652 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:56:04.116215 IP 10.0.2.1 > 224.0.0.22: igmp v3 report, 1 group record(s)
06:56:04.116483 IP6 fe80::200:ff:feaa:4 > ff02::16: HBH ICMP6, multicast listene
r report v2, 1 group record(s), length 28
06:56:04.116963 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:56:08.692804 IP 10.0.2.1 > 224.0.0.22: igmp v3 report, 1 group record(s)
06:56:11.796422 IP6 fe80::200:ff:feaa:4 > ff02::16: HBH ICMP6, multicast listene
r report v2, 1 group record(s), length 28
06:56:14.065861 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:56:14.117708 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:56:24.074886 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:56:24.118229 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:56:33.965365 ARP, Request who-has 10.0.2.1 tell 10.0.2.10, length 28
06:56:33.965405 ARP, Reply 10.0.2.1 is-at 00:00:00:aa:00:04, length 28
06:56:33.965408 IP 10.0.2.10 > 10.0.0.10: ICMP echo request, id 40, seq 1, lengt
h 40
06:56:33.965423 IP 10.0.2.1 > 10.0.2.10: ICMP time exceeded in-transit, length 6
8
06:56:33.965431 IP 10.0.2.10 > 10.0.0.10: ICMP echo request, id 40, seq 2, lengt
h 40
06:56:33.965436 IP 10.0.2.1 > 10.0.2.10: ICMP time exceeded in-transit, length 6
8
06:56:33.965439 IP 10.0.2.10 > 10.0.0.10: ICMP echo request, id 40, seq 3, lengt
h 40
```

Neste caso, n4 recebe e envia tráfego, até que o TTL (Time To Live) atinge o valor 0, altura tal, em que deixa de responder aos pedidos de tráfego. Daí o aparecimento de uma mensagem ICMP.

c. Qual deve ser o valor inicial mínimo do campo TTL para alcançar o destino 1? Verifique na prática que a sua resposta está correta.

O valor inicial mínimo do campo TTL deve ser de 1, pois se for 0, o pacote é descartado, e a mensagem ICMP “Time Exceeded” irá ocorrer de imediato.

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

O tempo é de 10 segundos por envio de tráfego.

```
06:58:44.039556 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:58:44.136584 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:58:54.054480 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:58:54.136714 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:59:04.072335 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:59:04.137658 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:59:14.097043 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:59:14.138809 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:59:24.064935 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:59:24.139872 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:59:34.059016 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:59:34.141037 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:59:44.109154 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:59:44.142168 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
06:59:54.070426 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
06:59:54.142705 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
07:00:04.120660 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
07:00:04.143862 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
07:00:14.084399 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
07:00:14.144932 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
07:00:24.055294 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
07:00:24.146017 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
07:00:34.091983 IP6 fe80::200:ff:feaa:4 > ff02::5: OSPFv3, Hello, length 36
07:00:34.147465 IP 10.0.2.1 > 224.0.0.5: OSPFv2, Hello, length 44
```

2.

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

O endereço IP da interface ativa do nosso computador é 192.168.100.150.

No.	Time	Source	Destination	Protocol	Length	Info
26	2.295450	192.168.100.150	193.136.9.240	ICMP	106	Echo (ping) request id=0x0001, seq=19/4864, ttl=1 (no response found!)

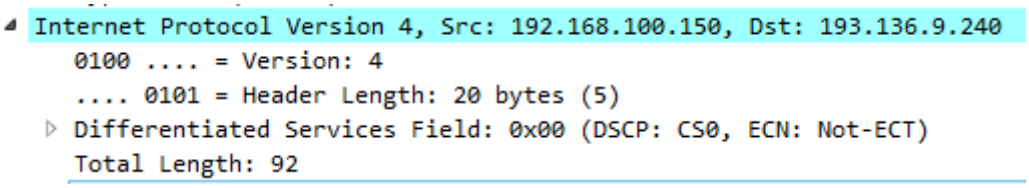
b. Qual é o valor do campo protocolo? O que identifica?

O valor do campo protocolo é ICMP(1) e este identifica o tipo de erro ocorrido, permitindo à fonte original alterar o seu comportamento de acordo com o erro relatado.

```
▷ Time to live: 1
  Protocol: ICMP (1)
```

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?

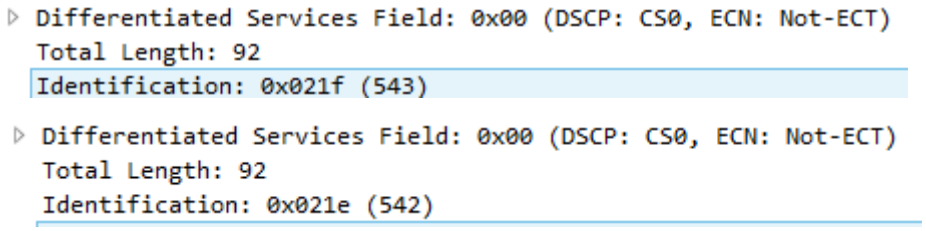
O cabeçalho IP(v4) tem 20 bytes enquanto que o campo de dados do datagrama tem 72. O resultado é obtido subtraindo o total de bytes do cabeçalho IP(v4) ao número total de bytes do datagrama que são 92.



d. O datagrama IP foi fragmentado? Justifique.

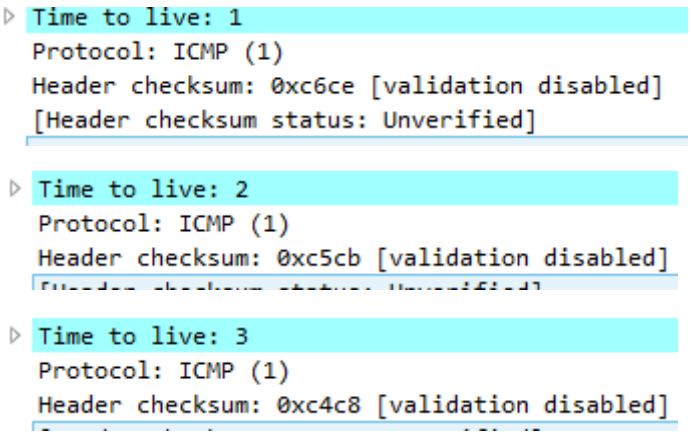
e. Ordene os pacotes capturados de acordo com o endereço IP fonte (e.g., selecionando o cabeçalho da coluna Source), e analise a sequência de tráfego ICMP gerado a partir do endereço IP atribuído à sua máquina. Para a sequência de mensagens ICMP enviadas pelo seu computador, indique que campos do cabeçalho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

O único campo que varia de pacote para pacote no cabeçalho IP(v4) são os bytes de identificação (ip.id)



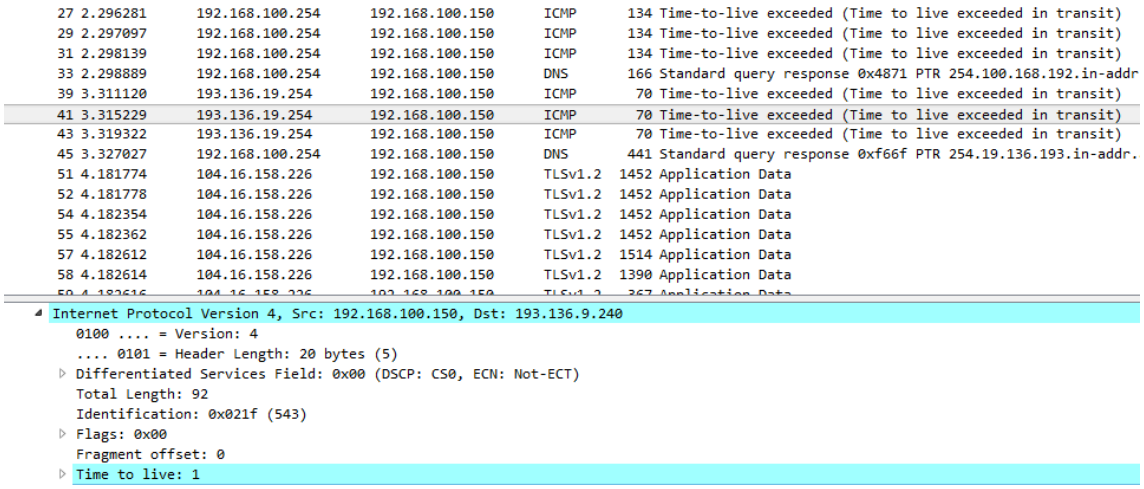
f. Observa algum padrão nos valores do campo de Identificação do datagrama IP e TTL?

Em relação ao TTL observamos que este aumenta de um em um a cada 3 mensagens ICMP enviadas pelo nosso computador. Quanto ao byte de identificação observamos que este aumenta um byte a cada mensagem ICMP enviada.



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 TTL é igual a 1 para todas as mensagens de resposta ICMP TTL exceeded isto porque o TTL não era suficiente para chegar ao destino, tendo que dar mais saltos para chegar ao mesmo. Como o TTL decresce em cada salto, seria de esperar que o valor do TTL de todas as mensagens ICMP TTL exceeded fosse 0, isto não acontece simplesmente porque o router sabe que no próximo salto teria que decrementar o TTL para 0 e o que faz é simplesmente produzir o erro antes, ou seja, quando o TTL ainda é 1.



3.

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

Ocorreu fragmentação do pacote, pois o MTU especificado era demasiado pequeno para que a mensagem inteira IPv4 passasse.

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

A parte do cabeçalho que indica que ocorreu fragmentação encontra-se na flag, mais especificamente no bit "More fragments", que se encontra a 1 (20 no byte do cabeçalho), indicando que este datagrama faz parte de uma mensagem IPv4 fragmentada.

Este datagrama faz parte do primeiro fragmento, pois o campo "Fragment Offset" encontra-se a 0.

```
▾ Flags: 0x01 (More Fragments)
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..1. .... = More fragments: Set
   Fragment offset: 0
```

O tamanho do datagrama é de 1500 bytes.

```
Total Length: 1500
```

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 campo que indica que isto não se trata do 1º fragmento é o "fragment offset", cujo valor é 1480. Há mais fragmentos, pois o bit na flag "More fragments" ainda se encontra com valor 1 (20 no byte do cabeçalho). Caso fosse o último fragmento, este teria a flag a 0.

```
▾ Flags: 0x01 (More Fragments)
      0... .... = Reserved bit: Not set
      .0.. .... = Don't fragment: Not set
      ..1. .... = More fragments: Set
   Fragment offset: 1480
```

d. Quantos fragmentos foram criados a partir do datagrama original? Como se deteta o último fragmento correspondente ao datagrama original?

Foram criados 3 fragmentos.

Tal como dito anteriormente, para detetar que nos encontramos no último fragmento, basta ver se a flag no "More fragments" se encontra a 0.

```
▾ [3 IPv4 Fragments (4008 bytes): #19(1480), #20(1480), #21(1048)]
      [Frame: 19, payload: 0-1479 (1480 bytes)]
      [Frame: 20, payload: 1480-2959 (1480 bytes)]
      [Frame: 21, payload: 2960-4007 (1048 bytes)]
      [Fragment count: 3]
      [Reassembled IPv4 length: 4008]
      [Reassembled IPv4 data: 0800fce70001001c2020202020202020202020202020...]
```

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.

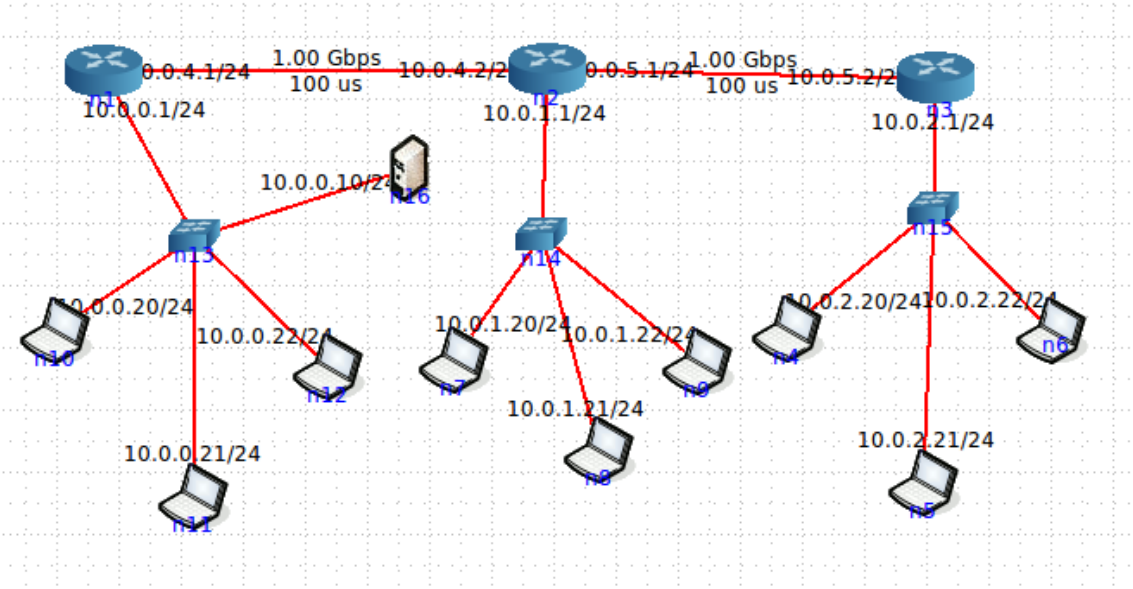
Temos o “Fragment offset”, que nos indica qual é a parte do datagrama total IPv4 a que corresponde o fragmento, e temos o campo “More fragments” na flag, que nos permite ver se o fragmento corresponde ao último (por exemplo, num primeiro fragmento o “fragment offset” a 0, e o “More fragments” a 1. Nos fragmentos seguintes, e antes do último, “More fragments” mantém-se a 1, e o “Fragment offset” vai aumentando de valor. No último fragmento, “More fragments” fica a 0, e “fragment offset” tem o valor mais alto).

Protocolo IPv4 (Parte II)

Endereçamento e Encaminhamento IP

1.

a. Indique que endereços IP e máscaras de rede foram atribuídas pelo CORE a cada equipamento. Se preferir, pode incluir uma imagem que ilustre de forma clara a topologia e o endereçamento.



b. Tratam-se de endereços públicos ou privados? Porquê?

Nesta topologia, todos os endereços IP são privados, porque as entidades a quem foram atribuídos pertencem todas a uma rede local. O próprio endereço IP associado a cada uma delas é prova disso mesmo. Todos eles foram atribuídos pelo CORE, seguindo um padrão 10.0.0.0, padrão associado à classe A dos endereços IP e reservada em particular para IP privados.

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

Não é atribuído nenhum endereço IP aos switches porque, ao estarem ligados a 3 laptops por departamento, vão mudar constantemente de pacote ethernet, consoante o laptop envolvido em dada comunicação na rede local.

d. Usando o comando ping certifique-se que existe conectividade IP entre os laptops dos utilizadores e o servidor do departamento A (basta verificar a conectividade de um laptop por departamento)

```
07:36:12.776658 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 27, seq 37, leng
th 64
07:36:12.776669 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 27, seq 37, length
 64
07:36:13.782662 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 27, seq 38, leng
th 64
07:36:13.782675 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 27, seq 38, length
 64
07:36:14.557824 IP 10.0.0.1 > 224.0.0.5: OSPFv2, Hello, length 44
07:36:14.725891 IP6 fe80::200:ff:feaa:9 > ff02::5: OSPFv3, Hello, length 36
07:36:14.782199 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 27, seq 39, leng
th 64
07:36:14.782214 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 27, seq 39, length
 64
07:36:15.787018 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 27, seq 40, leng
th 64
07:36:15.787029 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 27, seq 40, length
 64
07:36:16.799233 IP 10.0.0.20 > 10.0.0.10: ICMP echo request, id 27, seq 41, leng
th 64
07:36:16.799243 IP 10.0.0.10 > 10.0.0.20: ICMP echo reply, id 27, seq 41, length
 64
```

Laptop A

```
07:37:58.354900 IP 10.0.0.10 > 10.0.1.20: ICMP echo reply, id 26, seq 42, length
 64
07:37:59.373833 IP 10.0.1.20 > 10.0.0.10: ICMP echo request, id 26, seq 43, leng
th 64
07:37:59.373848 IP 10.0.0.10 > 10.0.1.20: ICMP echo reply, id 26, seq 43, length
 64
07:38:00.387037 IP 10.0.1.20 > 10.0.0.10: ICMP echo request, id 26, seq 44, leng
th 64
07:38:00.387051 IP 10.0.0.10 > 10.0.1.20: ICMP echo reply, id 26, seq 44, length
 64
07:38:01.399587 IP 10.0.1.20 > 10.0.0.10: ICMP echo request, id 26, seq 45, leng
th 64
07:38:01.399598 IP 10.0.0.10 > 10.0.1.20: ICMP echo reply, id 26, seq 45, length
 64
07:38:02.414281 IP 10.0.1.20 > 10.0.0.10: ICMP echo request, id 26, seq 46, leng
th 64
07:38:02.414295 IP 10.0.0.10 > 10.0.1.20: ICMP echo reply, id 26, seq 46, length
 64
07:38:03.426516 IP 10.0.1.20 > 10.0.0.10: ICMP echo request, id 26, seq 47, leng
th 64
07:38:03.426532 IP 10.0.0.10 > 10.0.1.20: ICMP echo reply, id 26, seq 47, length
 64
```

Laptop B

Laptop C

Ao fazer ping entre os diferentes laptops, houve resposta (como se pode ver nos prints). Logo, existe conectividade entre cada um dos laptop dos vários departamentos e o servidor do departamento A.

2.

a. Execute o comando netstart -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 da tabela. Se necessário consulte o manual respetivo (man netstat).

```
> n1 > 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        10.0.4.2        255.255.255.0   UG        0 0          0 eth2
10.0.2.0        10.0.4.2        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 eth2
10.0.5.0        10.0.4.2        255.255.255.0   UG        0 0          0 eth2

> n10 > netstat -rn:
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         10.0.0.1        0.0.0.0         UG        0 0          0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U         0 0          0 eth0
```

A coluna destination inclui os endereços IP para os quais o router ou o laptop enviaram pacotes. A coluna Gateway indica os endereços IP dos routers que fazer de intermediário entre a origem e o destino. A coluna Genmask indica a máscara da subrede. Neste caso, todas fazem referência ao destino do host. As flags definem o tipo de tráfego. Aqui, tanto o laptop como o router têm a flag U, que indica que o tráfego vai para cima, e alguns pacotes incluem a flag G, que indica que o router utiliza um gateway. A coluna MSS significa Maximum Segment Size, e diz qual o tamanho máximo do datagrama. A coluna Window aponta para o tamanho máximo de dados que o sistema aceita de uma só vez. A coluna irtt indica que rota está a ser usada. Na nossa topologia de rede, está a ser usada a rota por defeito (0). Por fim, a coluna Iface assinala a interface de rede usada para a transmissão/receção de pacotes.

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

Ao analisar os processos a correr no router n1, verificamos que corre o processo QUAGGA, que implementa OSPF (Open Shortest Path First), RIP (Routing Information Protocol) e BGP (Border Gateway Protocol), que usam encaminhamento dinâmico.

```
.pid -C /tmp/pycore.57557/n1.conf
root        58  0.0  0.1   3576  1120 ?        Ss   04:35   0:00 /usr/lib/quagga
/zebra -u root -g root -d
root        62  0.0  0.1   3992  1368 ?        Ss   04:35   0:00 /usr/lib/quagga
/ospf6d -u root -g root -d
root        63  0.0  0.1   4064  1392 ?        Ss   04:35   0:00 /usr/lib/quagga
/ospfd -u root -g root -d
root        92  0.0  0.0   2864  1016 ?        Rs   05:04   0:00 ps aux
```

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 localizado no departamento A. Use o comando* route delete *para o efeito. Que implicações tem esta medida para os utilizadores da empresa que acedem ao servidor. Justifique.*

```
> nl6 > 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 eth1
```

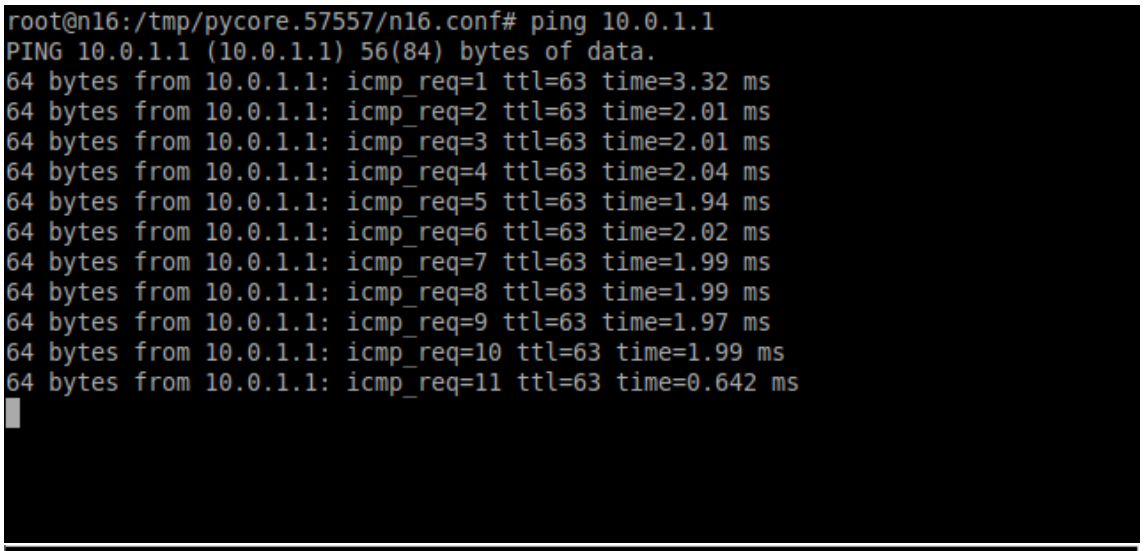
Se eliminarmos a rota por defeito, o servidor deixa de poder responder a outras redes, só podendo responder à sua própria rede.

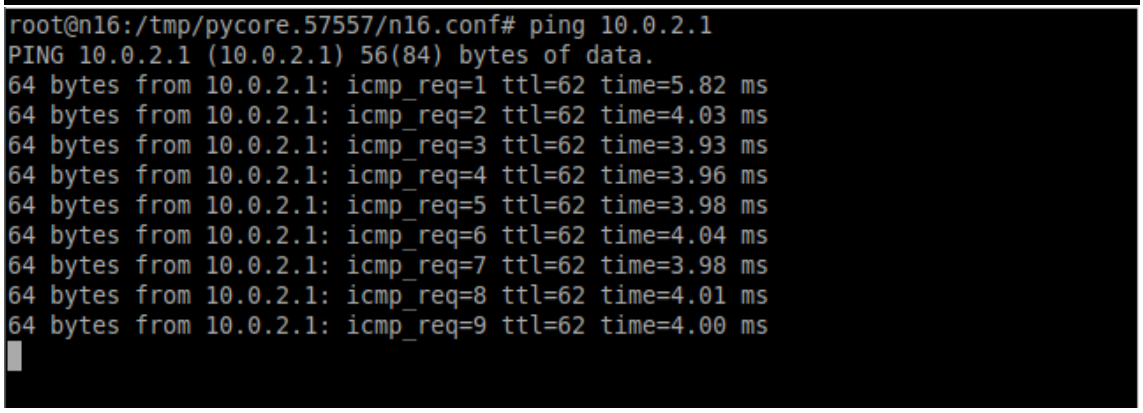
d. Adicione as rotas estáticas necessárias para restaurar a conectividade para o servidor, por forma a contornar a restrição imposta em c). Utilize para o efeito o comando route add *e registe os comandos que usou.*

```
route add -net 10.0.1.0 netmask 255.255.255.0 gw 10.0.0.1

route add -net 10.0.2.0 netmask 255.255.255.0 gw 10.0.0.1
```

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.*

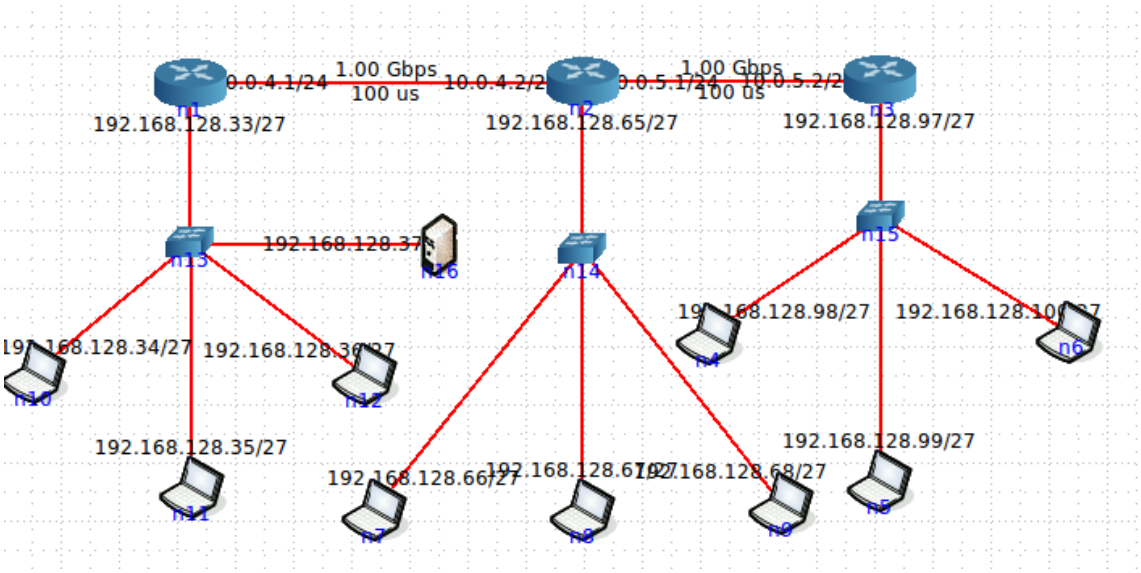




Como se pode ver, o comando ping indica que existe conexão entre os dois routers dos dois departamentos adjacentes.

3.

a. Assumindo que dispõe apenas de um único endereço de rede IP classe C 192.168.128.0/24, defina um novo esquema de endereçamento para as redes dos departamentos (mantendo a rede de core inalterada) e atribua endereços às interfaces dos vários sistemas envolvidos. Deve justificar as opções usadas.



Esta passa a ser o novo aspeto da topologia. Com os novos endereços IP atribuídos a cada uma das interfaces envolvidas em cada sub-rede.

- - - | - - - - -

(- representa 1 bit)

Sub-redes->2^3 - 2 = 6 sub-redes possíveis.

Hosts->2^5 - 2 = 30 Hosts por sub-rede.

Gama de Endereços IP válidos:

192.168.128.33...62/27 (Departamento A)

192.168.128.65...94/27 (Departamento B)

192.168.128.97...126/27 (Departamento C)

Trata-se de um esquema onde 3 dos 8 bits disponíveis vão ser usadas para as sub-redes e 5 bits para os hosts. A escolha dos endereços das interfaces em cada sub-rede (departamento) foi feita de modo a que os mesmos se encontrassem na mesma gama de valores do endereço do router do seu departamento.

Interfaces da sub-rede do Departamento C

(Router) n3->192.168.128.97/27

n4->192.168.128.98/27 <- Valor dentro da gama 97…126

n5->192.168.128.99/27 <- Valor dentro da gama 97...126

n6->192.168.128.100/27 <- Valor dentro da gama 97...126

Interfaces da sub-rede do Departamento B

(Router) n2->192.168.128.65/27

n7->192.168.128.66/27 <- Valor dentro da gama 65...94

n8->192.168.128.67/27 <- Valor dentro da gama 65...94

n9->192.168.128.68/27 <- Valor dentro da gama 65…94

Interfaces da Sub-rede do Departamento A

(Router) n1->192.168.128.33/27

n10->192.168.128.34/27 <- Valor dentro da gama 33…62

n11->192.168.128.35/27 <- Valor dentro da gama 33...62

n12->192.168.128.36/27 <- Valor dentro da gama 33...62

(Host) n16->192.168.128.37/27 <- Valor dentro da gama 33...62

b. Qual a máscara de rede que usou (em formato decimal)? Justifique.

Máscara de rede: 255.255.255.224 (/27, 27 bits a 1)

$1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1110\ 0000_2 = 1.1111111e+31_{10}$

Sendo o esquema de endereçamento por nós escolhido /27 (24 bits rede + 3 bits sub-rede) a máscara de rede será 27 bits a 1 e os 5 restantes, dos 32 totais, a 0 para representar o dito esquema por nós pretendido com sucesso.

c. Quantos hosts IP pode interligar em cada departamento? Justifique.

Com o esquema de endereçamento /27, estando 5 bits reservados para os hosts podemos interligar tantos hosts, quantos endereços diferentes tivermos, ou seja:

- - - | - - - - -

Hosts -> 2^5 – 2 (endereços reservados) = 30 endereços possíveis para atribuir aos hosts em cada sub-rede, ou seja, em cada departamento.

d. Garanta que conectividade IP entre as várias redes locais da empresa MIEInet é mantida

Foi testado se havia conectividade entre pelo menos um dos laptops de cada departamento e o servidor em A. Isso verificou-se, como se pode ver pelos resultados do comando ping, pelo que a conectividade entre departamentos foi mantida.

```
root@n10:/tmp/pycore.51705/n10.conf# ping 192.168.128.37
PING 192.168.128.37 (192.168.128.37) 56(84) bytes of data.
64 bytes from 192.168.128.37: icmp_req=1 ttl=64 time=0.063 ms
64 bytes from 192.168.128.37: icmp_req=2 ttl=64 time=0.030 ms
64 bytes from 192.168.128.37: icmp_req=3 ttl=64 time=0.030 ms
64 bytes from 192.168.128.37: icmp_req=4 ttl=64 time=0.032 ms
64 bytes from 192.168.128.37: icmp_req=5 ttl=64 time=0.038 ms
64 bytes from 192.168.128.37: icmp_req=6 ttl=64 time=0.032 ms
64 bytes from 192.168.128.37: icmp_req=7 ttl=64 time=0.032 ms
64 bytes from 192.168.128.37: icmp_req=8 ttl=64 time=0.050 ms
64 bytes from 192.168.128.37: icmp_req=9 ttl=64 time=0.038 ms
64 bytes from 192.168.128.37: icmp_req=10 ttl=64 time=0.035 ms
64 bytes from 192.168.128.37: icmp_req=11 ttl=64 time=0.038 ms
64 bytes from 192.168.128.37: icmp_req=12 ttl=64 time=0.035 ms
64 bytes from 192.168.128.37: icmp_req=13 ttl=64 time=0.036 ms
64 bytes from 192.168.128.37: icmp_req=14 ttl=64 time=0.062 ms
64 bytes from 192.168.128.37: icmp_req=15 ttl=64 time=0.035 ms
64 bytes from 192.168.128.37: icmp_req=16 ttl=64 time=0.040 ms
64 bytes from 192.168.128.37: icmp_req=17 ttl=64 time=0.072 ms
64 bytes from 192.168.128.37: icmp_req=18 ttl=64 time=0.037 ms
64 bytes from 192.168.128.37: icmp_req=19 ttl=64 time=0.041 ms
64 bytes from 192.168.128.37: icmp_req=20 ttl=64 time=0.036 ms
64 bytes from 192.168.128.37: icmp_req=21 ttl=64 time=0.035 ms
64 bytes from 192.168.128.37: icmp_req=22 ttl=64 time=0.033 ms
```

Ping do Laptop n10 (Departamento A) -> Host n16 (Departamento A)

```
09:04:38.436802 ARP, Reply 192.168.128.34 is-at 00:00:00:aa:00:00, length 28
09:04:39.432895 IP 192.168.128.34 > 192.168.128.37: ICMP echo request, id 26, se
q 39, length 64
09:04:39.432907 IP 192.168.128.37 > 192.168.128.34: ICMP echo reply, id 26, seq
39, length 64
09:04:40.433132 IP 192.168.128.34 > 192.168.128.37: ICMP echo request, id 26, se
q 40, length 64
09:04:40.433142 IP 192.168.128.37 > 192.168.128.34: ICMP echo reply, id 26, seq
40, length 64
09:04:40.949522 IP6 fe80::200:ff:feaa:9 > ff02::5: OSPFv3, Hello, length 36
09:04:41.104070 IP 192.168.128.33 > 224.0.0.5: OSPFv2, Hello, length 44
09:04:41.433139 IP 192.168.128.34 > 192.168.128.37: ICMP echo request, id 26, se
q 41, length 64
09:04:41.433156 IP 192.168.128.37 > 192.168.128.34: ICMP echo reply, id 26, seq
41, length 64
09:04:42.435002 IP 192.168.128.34 > 192.168.128.37: ICMP echo request, id 26, se
q 42, length 64
09:04:42.435013 IP 192.168.128.37 > 192.168.128.34: ICMP echo reply, id 26, seq
42, length 64
09:04:43.434001 IP 192.168.128.34 > 192.168.128.37: ICMP echo request, id 26, se
q 43, length 64
09:04:43.434013 IP 192.168.128.37 > 192.168.128.34: ICMP echo reply, id 26, seq
43, length 64
```

tcpdump Host n16 (Departamento A)

```
root@n7:/tmp/pycore.51705/n7.conf# ping 192.168.128.37
PING 192.168.128.37 (192.168.128.37) 56(84) bytes of data.
64 bytes from 192.168.128.37: icmp_req=1 ttl=62 time=1.84 ms
64 bytes from 192.168.128.37: icmp_req=2 ttl=62 time=1.07 ms
64 bytes from 192.168.128.37: icmp_req=3 ttl=62 time=1.93 ms
64 bytes from 192.168.128.37: icmp_req=4 ttl=62 time=1.98 ms
64 bytes from 192.168.128.37: icmp_req=5 ttl=62 time=1.99 ms
64 bytes from 192.168.128.37: icmp_req=6 ttl=62 time=2.06 ms
64 bytes from 192.168.128.37: icmp_req=7 ttl=62 time=1.98 ms
64 bytes from 192.168.128.37: icmp_req=8 ttl=62 time=1.98 ms
64 bytes from 192.168.128.37: icmp_req=9 ttl=62 time=1.17 ms
64 bytes from 192.168.128.37: icmp_req=10 ttl=62 time=1.18 ms
64 bytes from 192.168.128.37: icmp_req=11 ttl=62 time=1.21 ms
64 bytes from 192.168.128.37: icmp_req=12 ttl=62 time=0.296 ms
64 bytes from 192.168.128.37: icmp_req=13 ttl=62 time=0.865 ms
64 bytes from 192.168.128.37: icmp_req=14 ttl=62 time=0.301 ms
64 bytes from 192.168.128.37: icmp_req=15 ttl=62 time=1.14 ms
64 bytes from 192.168.128.37: icmp_req=16 ttl=62 time=1.15 ms
64 bytes from 192.168.128.37: icmp_req=17 ttl=62 time=1.10 ms
64 bytes from 192.168.128.37: icmp_req=18 ttl=62 time=1.13 ms
64 bytes from 192.168.128.37: icmp_req=19 ttl=62 time=1.13 ms
64 bytes from 192.168.128.37: icmp_req=20 ttl=62 time=1.17 ms
64 bytes from 192.168.128.37: icmp_req=21 ttl=62 time=1.11 ms
64 bytes from 192.168.128.37: icmp_req=22 ttl=62 time=1.03 ms
```

Ping do Laptop n7 (Departamento B) -> Host n16 (Departamento A)

```
15, length 64
09:06:19.491789 IP 192.168.128.66 > 192.168.128.37: ICMP echo request, id 26, se
q 16, length 64
09:06:19.491802 IP 192.168.128.37 > 192.168.128.66: ICMP echo reply, id 26, seq
16, length 64
09:06:20.493483 IP 192.168.128.66 > 192.168.128.37: ICMP echo request, id 26, se
q 17, length 64
09:06:20.493494 IP 192.168.128.37 > 192.168.128.66: ICMP echo reply, id 26, seq
17, length 64
09:06:20.885713 IP6 fe80::200:ff:feaa:9 > ff02::5: OSPFv3, Hello, length 36
09:06:21.116242 IP 192.168.128.33 > 224.0.0.5: OSPFv2, Hello, length 44
09:06:21.494994 IP 192.168.128.66 > 192.168.128.37: ICMP echo request, id 26, se
q 18, length 64
09:06:21.495006 IP 192.168.128.37 > 192.168.128.66: ICMP echo reply, id 26, seq
18, length 64
09:06:22.497080 IP 192.168.128.66 > 192.168.128.37: ICMP echo request, id 26, se
q 19, length 64
09:06:22.497091 IP 192.168.128.37 > 192.168.128.66: ICMP echo reply, id 26, seq
19, length 64
09:06:23.499049 IP 192.168.128.66 > 192.168.128.37: ICMP echo request, id 26, se
q 20, length 64
09:06:23.499059 IP 192.168.128.37 > 192.168.128.66: ICMP echo reply, id 26, seq
20, length 64
```

tcpdump Host n16 (Departamento A)

```
root@n6:/tmp/pycore.51706/n6.conf# ping 192.168.128.37
PING 192.168.128.37 (192.168.128.37) 56(84) bytes of data.
64 bytes from 192.168.128.37: icmp_req=1 ttl=61 time=2.75 ms
64 bytes from 192.168.128.37: icmp_req=2 ttl=61 time=0.802 ms
64 bytes from 192.168.128.37: icmp_req=3 ttl=61 time=4.04 ms
64 bytes from 192.168.128.37: icmp_req=4 ttl=61 time=1.14 ms
64 bytes from 192.168.128.37: icmp_req=5 ttl=61 time=4.03 ms
64 bytes from 192.168.128.37: icmp_req=6 ttl=61 time=3.45 ms
64 bytes from 192.168.128.37: icmp_req=7 ttl=61 time=3.99 ms
64 bytes from 192.168.128.37: icmp_req=8 ttl=61 time=4.01 ms
```

Ping do Laptop n6 (Departamento C) -> Host n16 (Departamento A)

```
09:24:25.719348 IP 192.168.128.100 > 192.168.128.37: ICMP echo request, id 26, s
eq 43, length 64
09:24:25.719358 IP 192.168.128.37 > 192.168.128.100: ICMP echo reply, id 26, seq
 43, length 64
09:24:26.720982 IP 192.168.128.100 > 192.168.128.37: ICMP echo request, id 26, s
eq 44, length 64
09:24:26.720991 IP 192.168.128.37 > 192.168.128.100: ICMP echo reply, id 26, seq
 44, length 64
09:24:27.708370 IP 192.168.128.33 > 224.0.0.5: OSPFv2, Hello, length 44
09:24:27.722469 IP 192.168.128.100 > 192.168.128.37: ICMP echo request, id 26, s
eq 45, length 64
09:24:27.722479 IP 192.168.128.37 > 192.168.128.100: ICMP echo reply, id 26, seq
 45, length 64
09:24:28.042014 IP6 fe80::200:ff:feaa:9 > ff02::5: OSPFv3, Hello, length 36
09:24:28.725266 IP 192.168.128.100 > 192.168.128.37: ICMP echo request, id 26, s
eq 46, length 64
09:24:28.725277 IP 192.168.128.37 > 192.168.128.100: ICMP echo reply, id 26, seq
 46, length 64
09:24:29.727177 IP 192.168.128.100 > 192.168.128.37: ICMP echo request, id 26, s
eq 47, length 64
09:24:29.727188 IP 192.168.128.37 > 192.168.128.100: ICMP echo reply, id 26, seq
 47, length 64
```

tcpdump Host n16 (Departamento A)

4.
Conclusão.

Após a realização deste trabalho o nosso grupo concluí que alargamos substancialmente o nosso conhecimento sobre o Protocolo IPv4, incluindo nomeadamente o estudo do formato de um pacote ou datagrama IP, endereçamento IP, encaminhamento IP e fragmentação de pacotes IP

Conseguimos na 1ª parte aprender a registar datagramas IP enviados e recebidos através da execução do programa traceroute, que nos permitiu aprender a descobrir a rota de pacotes desde uma origem IP até um determinado destino, assim como a entender o conceito de TTL (tanto Time-To-Live como o TTL exceeded). Conseguimos também perceber o objetivo das mensagens ICMP bem como a analisar os vários campos de um datagrama IP e a detalhar o processo de fragmentação realizado pelo IP.

Na 2ª parte, tivemos a nossa primeira abordagem a um sistema de redes locais, com uma dimensão significativa para o que nos era familiar até então. Dada a moderada dimensão desta dita rede, foi necessário que fossemos capazes de dominar conceitos tão variados, como por exemplo, subnetting e interpretação de mensagens ICMP, de modo a otimizar ao máximo a rede em questão, percebendo o que essas mensagens significam, e percebendo como podemos manipular os endereços IP em determinada topologia a nosso favor para uma melhor organização sem prejudicar, ou danificar o bom funcionamento e interação de todas as componentes do sistema.

Em suma, concluímos que tanto este como todos os outros 3 trabalhos permitiram-nos aumentar profundamente os nossos conhecimentos em vários aspetos relacionados com Redes de Computadores, conhecimentos esses que serão sem dúvida imprescindíveis para o futuro.

PL2.8

A75209 – João Almeida

A74806 – João Amorim

A75364 – João Araújo