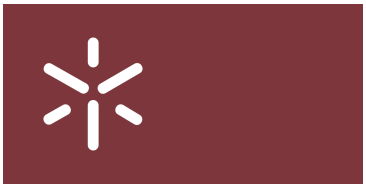


**Universidade do Minho**
Escola de Engenharia
Licenciatura em Engenharia Informática

# Redes de Computadores

Ano Letivo de 2024/2025

# Trabalho Prático TP2 - Grupo 50

**João Delgado** A106836
**Simão Mendes** A106928
**Nelson Rocha** A106884

Março, 2025

# Índice

# Lista de Figuras



# Lista de Tabelas

# 1. Introdução

## 1.1 Descrição

Este projeto tem como objetivo a exploração do Protocolo ***Internet Protocol* (IP)**, um componente essencial da comunicação na internet. Através de uma análise detalhada, procuraremos compreender a estrutura dos datagramas IP, o processo de fragmentação de pacotes, o endereçamento e o encaminhamento de dados na rede. Para isso, realizaremos atividades práticas utilizando o programa ***traceroute***, o que nos permitirá registrar e observar o comportamento dos datagramas em tempo real. Além disso, utilizaremos o **CORE** para simular redes e o ***Wireshark*** para capturar e analisar o tráfego de dados.

## 1.2 Motivação

A crescente dependência da tecnologia e da internet nas nossas vidas diárias torna essencial a compreensão dos protocolos que sustentam esta infraestrutura. O *Internet Protocol* (IP) é fundamental para a comunicação entre dispositivos, e entender o seu funcionamento é crucial para qualquer profissional da área de redes e tecnologia da informação. Este projeto não apenas nos permitirá explorar conceitos teóricos, mas também nos proporcionará experiências práticas que reforçarão o nosso conhecimento.

# 2. Parte 1

## 2.1 Questão 1

### 2.1.1: Problema Geral

O objetivo desta tarefa foi preparar uma topologia **CORE** de modo a verificar o comportamento da transmissão de pacotes utilizando do comando *traceroute*. O sistema é apresentado de seguida:

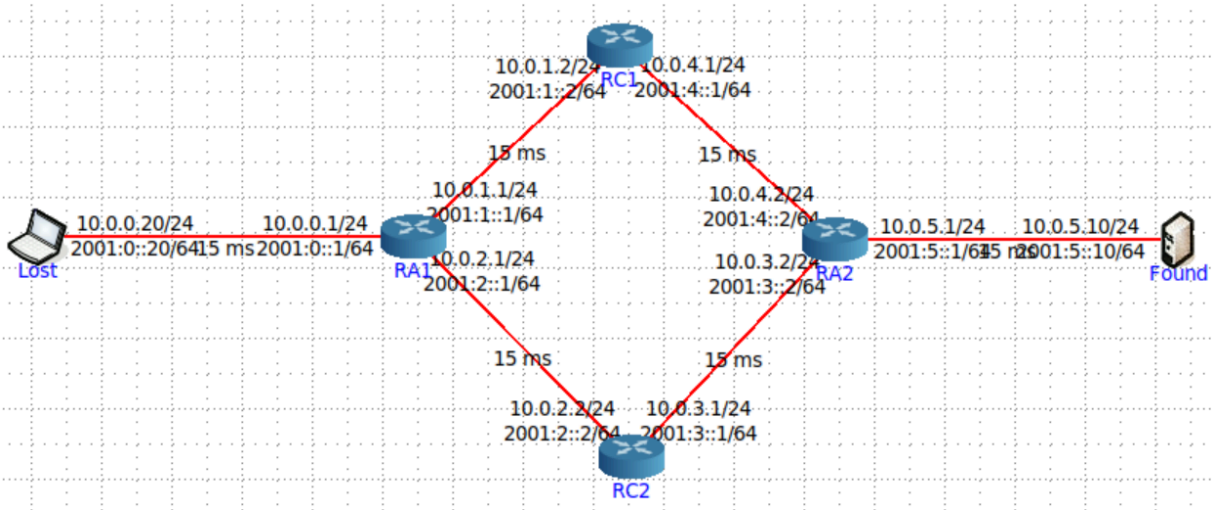



Figura 1: Topologia CORE orquestrada.

Como pode ser observado na imagem, um *host* (PC) denominado ***Lost*** está conectado ao *router* **RA1**, que, por sua vez, se conecta a dois *routers* intermédios, **RC1** e **RC2**. No lado oposto da figura, temos outro *host* (servidor) neste caso nomeado ***Found***, que está ligado ao *router* **RA2**, também conectado a **RC1** e **RC2**. Desta forma, existe uma rota estabelecida para a transmissão de pacotes entre *Lost* e *Found*, **permitindo a comunicação bidirecional**.

É importante ressaltar que cada ligação de rede possui um tempo de propagação fixo em **15 milissegundos**. Além disso, em cada conexão, é possível visualizar os endereços ***IPv4*** e ***IPv6*** de cada dispositivo nas extremidades da ligação.

### 2.1.2: Questão 1.a

**Problema:** Utilizando a ferramenta *Wireshark* e o comando *traceroute -I* para o endereço IP de *Lost*. Analise o tráfego ICMP enviado pelo sistema *Lost* e o tráfego recebido como resposta. Explique os resultados obtidos tendo em conta o princípio de funcionamento do *traceroute*.

**Resposta:** Antes de começar a responder propriamente à questão gostaríamos de esclarecer algumas noções sobre o funcionamento do comando *traceroute*. O *traceroute* rastreia a rota dos pacotes de dados até um destino, enviando pacotes com um tempo de vida (TTL) que começa em um e aumenta a cada iteração. Quando um *router* descarta um pacote, ele envia uma mensagem de volta, revelando o seu endereço. O processo continua até uma confirmação de chegada ao destino, revelando os *routers* intermediários e os tempos de resposta.

```bash
root@Lost:/tmp/pycore.45827/Lost.conf# traceroute -I 10.0.5.10
traceroute to 10.0.5.10 (10.0.5.10), 30 hops max, 60 byte packets
1 10.0.0.1 (10.0.0.1) 34.031 ms 33.395 ms 33.393 ms
```

```
4  2 10.0.1.2 (10.0.1.2) 67.685 ms 67.685 ms 67.683 ms
5  3 10.0.3.2 (10.0.3.2) 97.934 ms 97.934 ms 97.932 ms
6  4 10.0.5.10 (10.0.5.10) 130.430 ms 130.429 ms 130.427 ms
```

Como pode ser visto nesta execução do comando no terminal, os pacote finais enviados por *Lost* percorrem 3 dispositivos até à sua chegada no destino. Ao analisarmos os IP's de cada dispositivo, fica evidente que percurso realizado foi : *Lost* -> RA1 -> RC1 -> RA2 -> *Found*.

O *output* da captura dos pacotes do *Wireshark* encontra-se de seguida:

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 871 | 835.623092528 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=1/256, ttl=1 (no response found!) |
| 872 | 835.623110507 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=2/512, ttl=1 (no response found!) |
| 873 | 835.623114484 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=3/768, ttl=1 (no response found!) |
| 874 | 835.623118250 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=4/1024, ttl=2 (no response found!) |
| 875 | 835.623121304 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=5/1280, ttl=2 (no response found!) |
| 876 | 835.623123979 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=6/1536, ttl=2 (no response found!) |
| 877 | 835.623127325 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=7/1792, ttl=3 (no response found!) |
| 878 | 835.623129998 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=8/2048, ttl=3 (no response found!) |
| 879 | 835.623132533 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=9/2304, ttl=3 (no response found!) |
| 880 | 835.623136188 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=10/2560, ttl=4 (reply in 905) |
| 881 | 835.623138713 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=11/2816, ttl=4 (reply in 906) |
| → 882 | 835.623141357 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=12/3072, ttl=4 (reply in 907) |
| 883 | 835.623144261 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=13/3328, ttl=5 (reply in 908) |
| 884 | 835.623146766 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=14/3584, ttl=5 (reply in 909) |
| 885 | 835.623149561 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=15/3840, ttl=5 (reply in 910) |
| 886 | 835.623152394 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=16/4096, ttl=6 (reply in 911) |
| 887 | 835.657070825 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 888 | 835.657082815 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 889 | 835.657084397 | 10.0.0.1 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 890 | 835.658762231 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=17/4352, ttl=6 (reply in 912) |
| 891 | 835.658777467 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=18/4608, ttl=6 (reply in 913) |
| 892 | 835.658783907 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=19/4864, ttl=7 (reply in 914) |
| 893 | 835.690795532 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 894 | 835.690803415 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 895 | 835.690805047 | 10.0.1.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 896 | 835.691253708 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=20/5120, ttl=7 (reply in 915) |
| 897 | 835.691263464 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=21/5376, ttl=7 (reply in 916) |
| 898 | 835.691267762 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=22/5632, ttl=8 (reply in 917) |
| 899 | 835.721048417 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 900 | 835.721061749 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 901 | 835.721063152 | 10.0.3.2 | 10.0.0.20 | ICMP | 102 | Time-to-live exceeded (Time to live exceeded in transit) |
| 902 | 835.721969638 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=23/5888, ttl=8 (reply in 918) |
| 903 | 835.721980206 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=24/6144, ttl=8 (reply in 919) |
| 904 | 835.721984363 | 10.0.0.20 | 10.0.5.10 | ICMP | 74 | Echo (ping) request id=0x0028, seq=25/6400, ttl=9 (reply in 920) |
| 905 | 835.753555989 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x0028, seq=10/2560, ttl=61 (request in 880) |
| 906 | 835.753565484 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x0028, seq=11/2816, ttl=61 (request in 881) |
| ← 907 | 835.753567187 | 10.0.5.10 | 10.0.0.20 | ICMP | 74 | Echo (ping) reply id=0x0028, seq=12/3072, ttl=61 (request in 882) |

Figura 2: Captura do *Wireshark* para Topologia CORE.

A partir da imagem pode-se visualizar a emissão contínua de pacotes com o TTL a aumentar progressivamente de 1 em 1, até receber a primeira *reply* do dispositivo *Found*, logo após emitir o pacote com TTL = 9. É curioso notar que até ao TTL = 3 temos notificações de *time to live exceeded*, o que significa que os pacotes foram descartados. A partir de TTL = 4 temos mensagens de resposta enviadas do *Found* para *Lost*. Note-se que são realizados três testes para cada TTL, exatamente como foi visto no *output* anterior.

### 2.1.3: Questão 1.b

**Problema:** Qual deve ser o valor inicial mínimo do campo TTL para alcançar o servidor *Found*? Esboce um esquema com o valor do campo TTL à chegada a cada um dos routers percorridos até ao *Found*.

**Resposta:** Tendo em conta que existem, no mínimo, três *routers* entre os dois *hosts*, sem qualquer outra informação, pode-se concluir que o TTL mínimo para alcançar o servidor *Found* terá de ser 4.

Pelo *traceroute* pode-se tirar a mesma conclusão devido ao facto de o *output* do percurso apenas revelar dispositivos até 4, sendo o quarto o próprio servidor. Por outro lado, no *Wireshark* é possível verificar que a primeira resposta, que não produz *time to live exceeded*, dá-se quando o TTL = 4. É interessante mencionar que enquanto a resposta de *Found* não retorna para *Lost*, o *traceroute* continua a enviar pacotes com o TTL a aumentar, neste caso até 9.

O esquema que representa o TTL à chegada de cada um dos *routers* e *host* pode ser representado pela seguinte tabela:

| Time to Live | RA1 | RC1 | RA2 | Found |
|---|---|---|---|---|
| 1 | 1 | N | N | N |
| 2 | 2 | 1 | N | N |
| 3 | 3 | 2 | 1 | N |
| 4 | 4 | 3 | 2 | 1 |

Tabela 1: TTL à entrada nos dispostivos, ordenados por ordem de chegada.

Como se pode constatar o TTL irá diminuir numa unidade a cada passagem nos dispositivos. É importante notar que a notação "N" representa onde os pacotes não chegam, pois, como se sabe, após a saída do dispositivo o TTL é diminuído numa unidade, sendo o pacote descartado caso o valor alcance zero.

#### 2.1.4: Questão 1.c

**Problema:** Calcule o valor médio do tempo de ida-e-volta (RTT - *Round-Trip Time*) obtido no acesso ao servidor.

**Resposta:** Após enviar cinco pacotes com TTL = 4, analisamos o *output* do *traceroute* e do *Wireshark* sobre o momento de emissão dos pacotes em *Lost* até à receção em *Found* o que permitiu obter os seguintes valores:

| Pacote | Origem (ms) | Destino (ms) | Tempo (ms) |
|---|---|---|---|
| 1 | 0.623136188 | 0.753555989 | 0.130419801 |
| 2 | 0.623138713 | 0.753565484 | 0.130426771 |
| 3 | 0.623141357 | 0.753567187 | 0.130425830 |
| 4 | 0.623144261 | 0.753568339 | 0.130424078 |
| 5 | 0.623146766 | 0.753570071 | 0.130423305 |
| Média | 0.623141457 | 0.753765214 | 0.130423957 |

Tabela 2: RTT obtido no acesso ao servidor.

Por conseguinte pode-se concluir que, em média, o RTT é 0.130423957 milissegundos, nesta configuração de rede.

#### 2.1.5: Questão 1.d

**Problema:** O valor médio do atraso num sentido (*One-Way Delay*) poderia ser calculado com precisão dividindo o RTT por dois? O que torna difícil o cálculo desta métrica numa rede real?

**Resposta:** A divisão do RTT por dois não é um método preciso para calcular o *One-Way Delay* devido a vários fatores. Primeiramente, a assimetria da rede pode resultar em tempos de ida e volta diferentes, pois as rotas e as condições de tráfego podem variar entre os dois sentidos. Além disso, o

congestionamento na rede pode causar atrasos distintos para pacotes que seguem direções opostas. O tempo de processamento em dispositivos intermediários também pode ser diferente para pacotes de ida e volta, contribuindo para a imprecisão.

Por fim, medir o *One-Way Delay* com precisão requer que os relógios dos dispositivos estejam sincronizados, o que pode ser difícil de alcançar. Estes fatores tornam o cálculo do *One-Way Delay* a partir do RTT uma estimativa imprecisa em redes reais, e, portanto, é preferível utilizar métodos diretos de medição do *One-Way Delay* para obter resultados mais confiáveis.

## 2.2 Questão 2

### 2.2.1: Problema Geral

Nesta tarefa foi novamente utilizado o *Wireshark* e o comando *traceroute*, sendo que o objetivo passou por selecionar e analisar uma mensagem com o protocolo ICMP obtida ao realizar *traceroute* para a máquina destino *marco.uminho.pt*. Mais concretamente nesta fase do trabalho pretende-se analisar o pacote no nível protocolar IP, em particular, no seu cabeçalho IP. O comando utilizado foi:

```bash
$ traceroute -I marco.uminho.pt $
```

Como se pode observar, não foi passado o tamanho do pacote como argumento, assim sendo, o tamanho do pacote pode ser variável.

A primeira mensagem ICMP capturada que foi selecionada encontra-se de seguida:

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 22 | 9.230973736 | 172.26.98.177 | 193.136.9.240 | ICMP | 74 | Echo (ping) request id=0xe965, seq=1/256, ttl=1 (no response found!) |

Figura 3: Captura da primeira mensagem ICMP - *Wireshark* .

### 2.2.2: Questão 2.a

**Problema:** Qual é o endereço IP da interface ativa do computador?

**Resposta:** Como pode ser visto no campo *Source* da Figura 3, o IP da interface ativa é **172.26.98.177**. Este endereço IP corresponde à máquina em que foi executado o comando *traceroute*, como era de se esperar. Assim, o IP da interface ativa é o ponto de partida para o envio dos pacotes na rede.

### 2.2.3: Questão 2.b

**Problema:** Qual é o valor do campo *Protocol*? O que permite identificar?

**Resposta:** O valor do campo *Protocol* é **ICMP**, que significa *Internet Control Message Protocol*. O ICMP é um protocolo fundamental no leque dos protocolos da rede, utilizado principalmente para enviar mensagens de controlo e erro entre dispositivos de rede. Este permite que os dispositivos comuniquem informações sobre o estado da rede, como a notificação de que um *host* não está acessível ou que um pacote não pôde ser entregue. Além disso, o ICMP é frequentemente utilizado em ferramentas de diagnóstico de rede, como o *traceroute*, utilizado no decorrer deste trabalho.

Neste caso em específico, o protocolo é utilizado para sinalizar à *Source* que o pacote de rede foi perdido, o que pode ocorrer durante a execução do comando *traceroute*. Esta notificação permite que a máquina de origem identifique, parcialmente, onde a perda ocorreu na rota até o destino.

#### 2.2.4: Questão 2.c

**Problema:** Quantos *bytes* tem o cabeçalho IPv4? Quantos *bytes* tem o campo de dados (*payload*) do datagrama? Como se calcula o tamanho do *payload*?

**Resposta:** Para responder a esta pergunta, é necessário ter uma visão mais aprofundada do protocolo IP e da estrutura do datagrama. Deste modo, segue-se a figura que representa os campos do datagrama do nosso pacote:

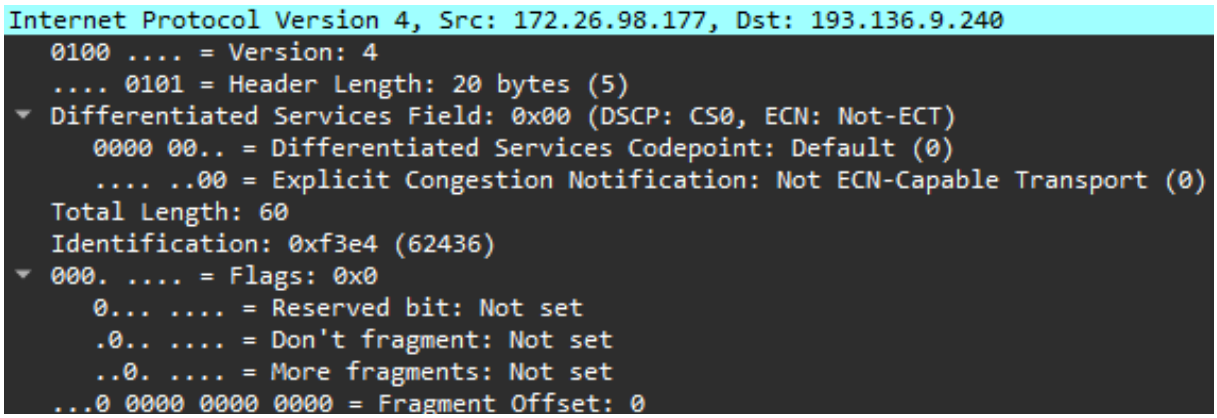



Figura 4: Captura da primeira mensagem ICMP - Datagrama IPv4 - *Wireshark*.

Certamente, a utilização do *Wireshark* foi fundamental para a análise realizada. Na captura, observamos que o tamanho do cabeçalho é de 20 *bytes* e o tamanho total do datagrama é de 60 *bytes*. Através da seguinte fórmula, podemos calcular o tamanho do *payload*:

**Tamanho do *Payload*** = Tamanho Total do Datagrama − Tamanho do Cabeçalho;

**Tamanho do *Payload*** $= 60 - 20 = 40$ *bytes*;

Portanto, o **Tamanho do *Payload*** é de 40 *bytes*.

#### 2.2.5: Questão 2.d

**Problema:** O datagrama IP foi fragmentado? Justifique.

**Resposta:** Antes de determinar se o datagrama foi fragmentado, é importante definir o que é a fragmentação. A fragmentação ocorre quando um datagrama IP é dividido em partes menores para ser transmitido em redes com um limite de tamanho de pacote (MTU).

Para verificar se o datagrama foi fragmentado devemos observar o cabeçalho IP. Se a *flag More Fragments* (MF) estiver ativa e/ou se o campo *Fragment Offset* for maior que zero, é certo que o datagrama foi fragmentado. Além disso, a *flag Don't Fragment* (DF) pode ser utilizada para indicar que o datagrama não deve ser fragmentado.

No caso específico do nosso datagrama, como pode ser visto na Figura 4, ambas as *flags More Fragments* e *Don't Fragment* não se encontram definidas (*Not Set*). Esta informação, juntamente com o facto de que o *Fragment Offset* é zero, indica que o datagrama IP não foi fragmentado.

#### 2.2.6: Questão 2.e

**Problema:** 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. Justifique estas mudanças.

**Resposta:** Na análise do tráfego ICMP gerado pela interface da máquina, alguns campos do cabeçalho IP variam entre pacotes:

***Identification:*** Cada pacote ICMP (como os *echo requests*) recebe um valor único neste campo para possibilitar a correta reconstrução caso o pacote seja fragmentado. Mesmo quando a fragmentação não ocorre, a variação deste campo é necessária para identificar de forma individual cada pacote enviado.

***Header Checksum:*** Recalculado a cada salto devido à alteração do TTL, garantindo a integridade dos dados.

***Time to Live (TTL):*** Decrementado a cada *router*, evitando *loops* e controlando a permanência dos pacotes na rede.

***Source e Destination IP:***

- ***Echo Request***: Origem é a máquina emissora e o destino é a testada.
- ***Echo Reply:*** Inversão dos papéis de origem e destino.
- **TTL Excedido:** O *router* envia *Time Exceeded* com o seu próprio IP como origem e o remetente original como destino.

***Stream Index:*** O *stream index* é útil para identificar a sequência de pacotes num fluxo de comunicação. Quando se observa que o *stream index* passa de 6 para 7, isso pode indicar que um novo fluxo de pacotes foi iniciado, como uma resposta ICMP *Time Exceeded* gerada quando o TTL de um pacote original chega a 0. Esta mudança é importante para rastrear a ordem dos pacotes e entender como as mensagens ICMP estão a ser geradas e respondidas na rede.

#### 2.2.7: Questão 2.f

**Problema:** Observa algum padrão nos valores do campo de Identificação do datagrama IP e TTL?

**Resposta:** A análise dos pacotes ICMP revela padrões distintos nos campos Identificação e TTL, o que facilita a diferenciação de mensagens e a gestão da transmissão.

***Echo Requests:*** O campo de identificação aumenta sequencialmente (ex.: 62436 a 62451), o que garante unicidade e auxilia na reconstrução de pacotes fragmentados.

**Outras Mensagens ICMP:** Respostas e erros (como TTL excedido) apresentam identificações distintas, sem sobreposição com os *echo requests* (ex.: 20566 a 20568, 6662 a 6664). Isto permite diferenciar mensagens e identificar as suas origens com precisão.

**Comportamento do TTL:**

- **Decremento a Cada Salto:** O TTL é reduzido em cada *router*, impedindo que pacotes circulem indefinidamente na rede.
- **Respostas Relacionadas ao TTL:** Se o TTL chega a 0 antes do destino, o pacote é descartado e um ICMP *Time Exceeded* é enviado. Caso o destino seja alcançado, a resposta ao *echo request* pode indicar a quantidade de saltos necessários.

**Conclusão:** Os valores de identificação seguem um padrão sequencial nos *echo requests*, enquanto os demais tipos de mensagens apresentam faixas de identificação distintas, garantindo a correta associação e tratamento dos pacotes. Já o TTL, ao ser decrementado a cada *router*, assegura que os pacotes não permanecem na rede indefinidamente e possibilita o diagnóstico das rotas percorridas na comunicação ICMP.

#### 2.2.8: Questão 2.g

**Pergunta:** Ordene o tráfego capturado por endereço destino e encontre a série de respostas ICMP *TTL Exceeded* enviadas ao seu computador.

- **Alínea i.:** Qual é o valor do campo TTL recebido no seu computador? Este valor permanece constante para todas as mensagens de resposta ICMP *TTL Exceeded* recebidas no seu computador? Porquê?

**Resposta:** Ao analisar as mensagens ICMP *TTL Exceeded* recebidas, observou-se que:

- As três primeiras mensagens apresentam TTL = 255.
- As três seguintes apresentam TTL = 254.
- As últimas três apresentam TTL = 253.

Isto demonstra que o valor do TTL recebido não permanece constante em todas as mensagens. O motivo para essa variação é o seguinte: quando um pacote atinge um *router* com o TTL esgotado, o *router* descarta o pacote original e gera uma mensagem ICMP *TTL Exceeded*. Este *router* define um valor inicial alto (por exemplo, 255) para essa mensagem, a fim de garantir que ela consiga retornar ao remetente. Contudo, durante o trajeto de volta, cada *router* intermediário decresce o valor do TTL numa unidade, fazendo com que o valor final recebido (255, 254, 253, etc.) reflita o número de saltos percorridos.

- **Alínea ii.:** Por que razão as mensagens de resposta ICMP *TTL Exceeded* são sempre enviadas na origem com um valor relativamente alto?

**Resposta:** As mensagens ICMP *TTL Exceeded* são sempre enviadas com um valor relativamente alto (como 255) na origem para assegurar que consigam alcançar o remetente original. Este valor elevado é configurado pelo *router* que gera a mensagem para compensar os decrementos que ocorrerão ao longo dos diversos saltos de retorno. Assim, mesmo que o TTL seja reduzido em cada *router* pelo qual a mensagem passa, o valor inicial alto garante que ela não seja descartada antes de chegar ao destino, permitindo que o remetente seja devidamente informado sobre a insuficiência do TTL.

#### 2.2.8: Questão 2.h

**Problema:** A informação contida no cabeçalho ICMP poderia ser incluída no cabeçalho IPv4? Se sim, quais seriam as suas vantagens/desvantagens?

**Resposta:** Embora seja teoricamente possível incluir os campos do ICMP no cabeçalho IPv4, essa integração não é adotada na prática. Entre os possíveis **pontos positivos**, destaca-se:

**Redução de *Overhead*:** Poderia haver uma pequena economia de *bytes* e uma simplificação no processamento, já que seria necessário analisar apenas um cabeçalho.

Porém, **as desvantagens** superam esta vantagem, pois:

**Perda de Modularidade:** Manter o IPv4 separado do ICMP permite que cada protocolo evolua e seja atualizado de forma independente, sem interferir um no outro.

**Flexibilidade Reduzida:** O ICMP precisa lidar com diversos tipos de mensagens (como *Echo*, *TTL Exceeded*, *Destination Unreachable*), o que exigiria campos específicos. A integração no IPv4 dificultaria a adaptação e expansão para novos tipos de mensagens.

**Problemas de Compatibilidade:** Alterar o cabeçalho IPv4 para incluir informações de ICMP modificaria um padrão amplamente estabelecido, o que poderia gerar dificuldades de interoperabilidade entre diferentes dispositivos e *softwares*.

Em resumo, apesar da viabilidade teórica, as desvantagens em termos de modularidade, flexibilidade e compatibilidade justificam a manutenção dos cabeçalhos separados para IPv4 e ICMP.

## 2.3 Questão 3

### 2.3.1: Problema Geral

Pretende-se agora analisar a fragmentação de pacotes IP. Usando o *Wireshark*, capturamos e observamos o tráfego gerado depois do tamanho de pacote ter sido definido para 3850 ($3800 + X$, onde $X$ é o número do grupo, neste caso 50). De modo a poder visualizar os fragmentos, desativamos a opção *Reassemble fragmented IPv4 datagrams*.

### 2.3.2: Questão 3.a

**Problema:** Após a localização da primeira mensagem ICMP. Coloca-se a questão de porque é que houve necessidade de fragmentar o pacote inicial?

**Resposta:** O pacote ICMP original era maior que o MTU *(Maximum Transmission Unit)* da rede. O MTU padrão para *Ethernet* é 1500 *bytes*, e como o pacote foi definido para $3800 + 50$ *bytes*, ele ultrapassou este limite. O *router* intermediário, ao perceber que o pacote é maior que o MTU da *interface*, dividiu-o em fragmentos menores para que pudesse ser transmitido pela rede. Cada fragmento possui um *offset* indicando a sua posição no pacote original.

### 2.3.3: Questão 3.b

**Problema:** Ao imprimir o primeiro fragmento do datagrama IP segmentado, três questões surgiram:

**1.** Que informação no cabeçalho indica que o datagrama foi fragmentado?

**2.** Que informação no cabeçalho IP indica que se trata do primeiro fragmento?

**3.** Qual é o tamanho deste datagrama IP?

**Resposta:** Analisando o primeiro pacote (primeiro fragmento) no *Wireshark*, podemos responder às questões desta alínea da seguinte forma:

**1.** O campo *Flags* do cabeçalho IP apresenta o *bit* **MF** ***(More Fragments)*** definido para 1 (por vezes aparece em hexadecimal como 0x2000 ou explicitamente *More Fragments: Set*). Além disso, o próprio *Wireshark* pode mostrar a indicação ***Fragmented IP datagram*** ou *More fragments: Set* no detalhamento do protocolo IP.

**2.** O campo ***Fragment Offset*** é igual a 0 no primeiro fragmento, indicando que este é o início do datagrama antes da fragmentação.

**3.** Pelo campo ***Total Length*** do cabeçalho IP, vemos que o tamanho do primeiro fragmento é 1500 *bytes*.

Observe que este valor corresponde ao tamanho total do fragmento (incluindo o cabeçalho IP), não ao tamanho original completo do datagrama antes da fragmentação.

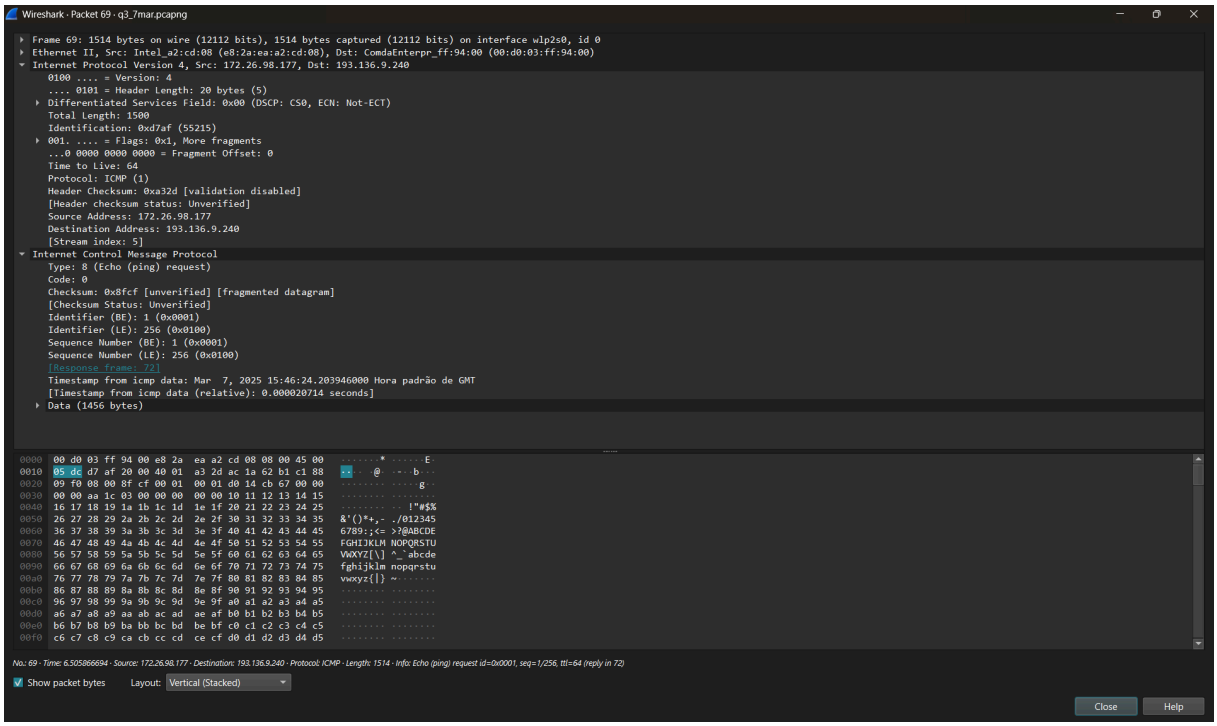



Figura 5: Informação sobre o Primeiro Fragmento do datagrama IP Segmentado.

### 2.3.4: Questão 3.c

**Problema:** Ao imprimir o segundo fragmento do datagrama IP original, colocaram-se três questões:

**1.** Que informação do cabeçalho IP indica que não se trata do 1º fragmento?

**2.** Existem mais fragmentos?

**3.** O que nos permite afirmar isso?

**Resposta:**

**1.** O campo ***More Fragments*** no primeiro fragmento deve ser 1, mas como neste caso é 0, referimo-nos ao segundo fragmento.

**2.** Como o campo ***More Fragments*** é igual a 0 no segundo fragmento, este será o segundo e último.

**3.** O pacote original foi dividido em dois fragmentos, sendo que o último indica o fim da fragmentação. O tamanho do pacote original é a soma dos dados dos dois fragmentos. A rede tem um **MTU menor** que o tamanho do pacote, causando a fragmentação.

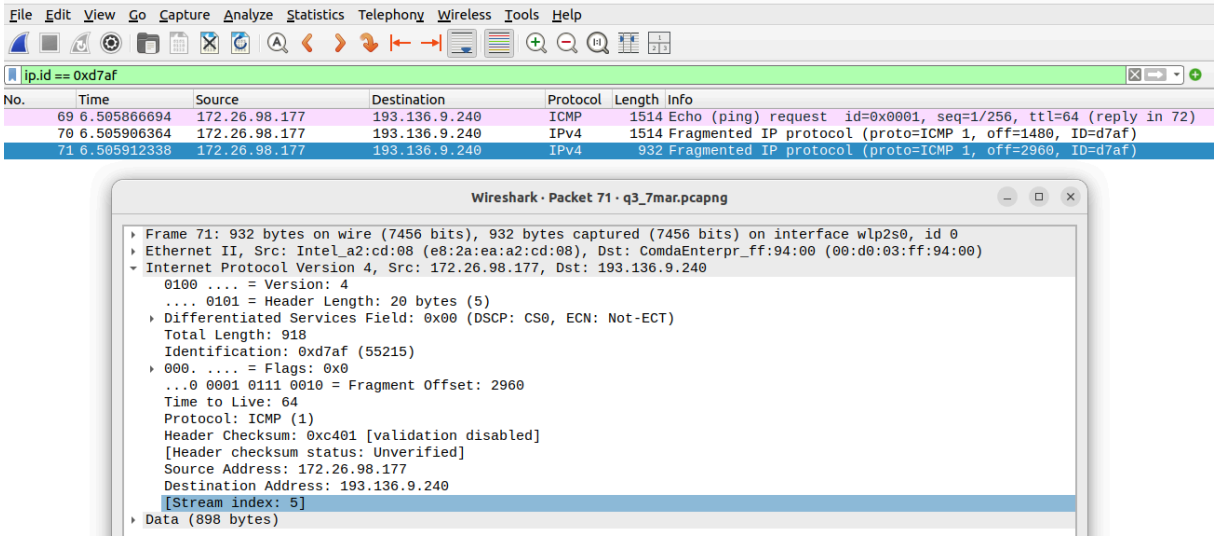



Figura 6: Informação sobre o Segundo Fragmento do datagrama IP Segmentado.

### 2.3.5: Questão 3.d

**Problema:** Continuando com a análise do segundo fragmento do datagrama IP também é possível responder às seguintes questões:

**1.** Quantos fragmentos foram criados a partir do datagrama original?

**2.** Como se detecta o último fragmento correspondente ao datagrama original?

**3.** Estabeleça um filtro no *Wireshark* que permita listar apenas o último fragmento do primeiro datagrama IP segmentado.

**Resposta:**

**1.** Pelo que foi possível concluir na análise da alínea anterior, foram criados dois fragmentos a partir do datagrama original.

**2.** O último fragmento do datagrama original será obtido através da análise do campo ***More Fragments***. Quando este estiver a 0, não haverá mais nenhum fragmento seguinte, fazendo deste fragmento o último.

**3.** O filtro pode ser expresso da seguinte forma:

```
1  ip.id == 0xd7af && ip.flags.mf == 0                                    Wireshark
```

É importante notar que o valor que segue o ***ip.id*** deve ser substituído pelo identificador desejado.

### 2.3.6: Questão 3.e

**Problema:** Agora passaremos à análise de quais são os campos que mudam no cabeçalho IP entre os diferentes fragmentos, explicando de que forma essa informação nos permite reconstruir o datagrama original.

**Resposta:** Os campos que mudam entre os fragmentos são ***More Fragments***, que indica se há mais fragmentos, e ***Fragment Offset***, que define a posição dos dados no pacote original. Já o campo ***Identification*** permanece o mesmo para todos os fragmentos do mesmo pacote, permitindo agrupá--los e reconstruir o datagrama original na ordem correta.

#### 2.3.7: Questão 3.f

**Problema:** Após todas estas conclusões e análises, conseguimos estimar, teoricamente, o número de fragmentos gerados e o número de *bytes* transportados em cada um dos fragmentos.

**Resposta:** Tal como é dito no enunciado, o pacote terá $3800 + 50$ *bytes*, uma vez que o nosso grupo é o grupo PL50, logo o pacote terá 3850 *bytes*. Ao analisar o tamanho dos dados do primeiro fragmento é possível concluir que cada fragmento terá no máximo 1480 *bytes*. Logo, teoricamente, existirão 2 fragmentos de 1480 *bytes* e um de 890 *bytes*. É importante notar que estes valores poderão ser diferentes, mas relativamente perto dos valores teoricamente calculados. Cada fragmento será definido pelos campos: ***More Fragments***, ***Fragment Offset***, ***Identification*** e o campo ***Data***.

#### 2.3.8: Questão 3.g

**Problema:** Tendo em conta o conceito de Fragmentação apresentado nas aulas teóricas conseguimos compreender qual a razão de apenas o primeiro fragmento de cada pacote ser identificado pelo *Wireshark* como sendo um pacote ICMP.

**Resposta:** Apenas o **primeiro fragmento** é identificado como ICMP porque ele contém o **cabeçalho ICMP**, que é necessário para o *Wireshark* reconhecer o protocolo de camada superior. Os **fragmentos subsequentes** contêm apenas partes dos dados e, portanto, são tratados como fragmentos IP genéricos. Isto está alinhado com o conceito de fragmentação IP, onde apenas o primeiro fragmento carrega as informações completas do protocolo de camada superior.

#### 2.3.9: Questão 3.h

**Problema:** Neste ponto são colocadas duas questões fundamentais na compreensão da Fragmentação:

**1.** Com que valor é o tamanho do datagrama comparado, a fim de se determinar se este deve ser fragmentado?

**2.** Quais seriam os efeitos na rede ao aumentar/diminuir este valor?

**Resposta:**

**1.** Para determinar se um datagrama IP deve ser fragmentado, o seu tamanho é comparado ao MTU da rede, que, neste caso, é 1514 *bytes*. Normalmente, o MTU é 1500 *bytes*, logo o MTU existente está próximo. Esta diferença pode ser justificada pela tecnologia de rede utilizada.

**2.** Ao afetar este valor, iria ser afetada a fragmentação do datagrama, levando a um conjunto de alterações no desempenho e na compatibilidade.

Se o valor aumentasse, haveria uma menor fragmentação e maior eficiência, mas aumentariam os problemas com compatibilidade e a latência em redes congestionadas.

Por sua vez, se o valor diminuísse, haveria maior compatibilidade, menor perda de pacotes e uma latência reduzida. Contudo, isto traria um aumento de fragmentação, desempenho reduzido e um aumento do *Overhead*.

#### 2.3.10: Questão 3.i

**Problema:** Agora, através das funcionalidades do *Linux* é possível determinar o valor máximo de *SIZE* sem que ocorra fragmentação do pacote. Determine o valor de *SIZE*.

**Resposta:** Para realizar o cálculo do *SIZE* devemos fazer a subtração do MTU com o cabeçalho do IP e com o cabeçalho ICMP. Assim sendo, o cálculo do *SIZE* ficaria: SIZE $= 1500 - 20 - 8 = 1472$ *bytes*.

```bash
nelson@nelson-TECRA-Z50-A:~$ ping -M do -s 1472 marco.uminho.pt
PING marco.uminho.pt (193.136.9.240) 1472(1500) bytes of data.
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=1 ttl=61 time=12.7 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=2 ttl=61 time=12.4 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=3 ttl=61 time=15.7 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=4 ttl=61 time=19.4 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=5 ttl=61 time=13.2 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=6 ttl=61 time=14.3 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=7 ttl=61 time=14.4 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=8 ttl=61 time=11.8 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=9 ttl=61 time=10.5 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=10 ttl=61 time=17.2 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=11 ttl=61 time=14.5 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=12 ttl=61 time=9.63 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=13 ttl=61 time=9.52 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=14 ttl=61 time=8.10 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=15 ttl=61 time=4.31 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=16 ttl=61 time=10.0 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=17 ttl=61 time=14.7 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=18 ttl=61 time=125 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=19 ttl=61 time=12.5 ms
1480 bytes from marco.uminho.pt (193.136.9.240): icmp_seq=20 ttl=61 time=87.4 ms
```

Como se pode concluir através do *output* do terminal acima, o destino respondeu com pacotes de 1480 *bytes*, que incluem o *payload* de 1472 *bytes* + 8 *bytes* do cabeçalho ICMP. Com isto, podemos afirmar que não será necessário fazer a fragmentação, logo o pacote será enviado por inteiro.

# 3. Parte 2

A segunda parte deste trabalho incide na seguinte topologia:

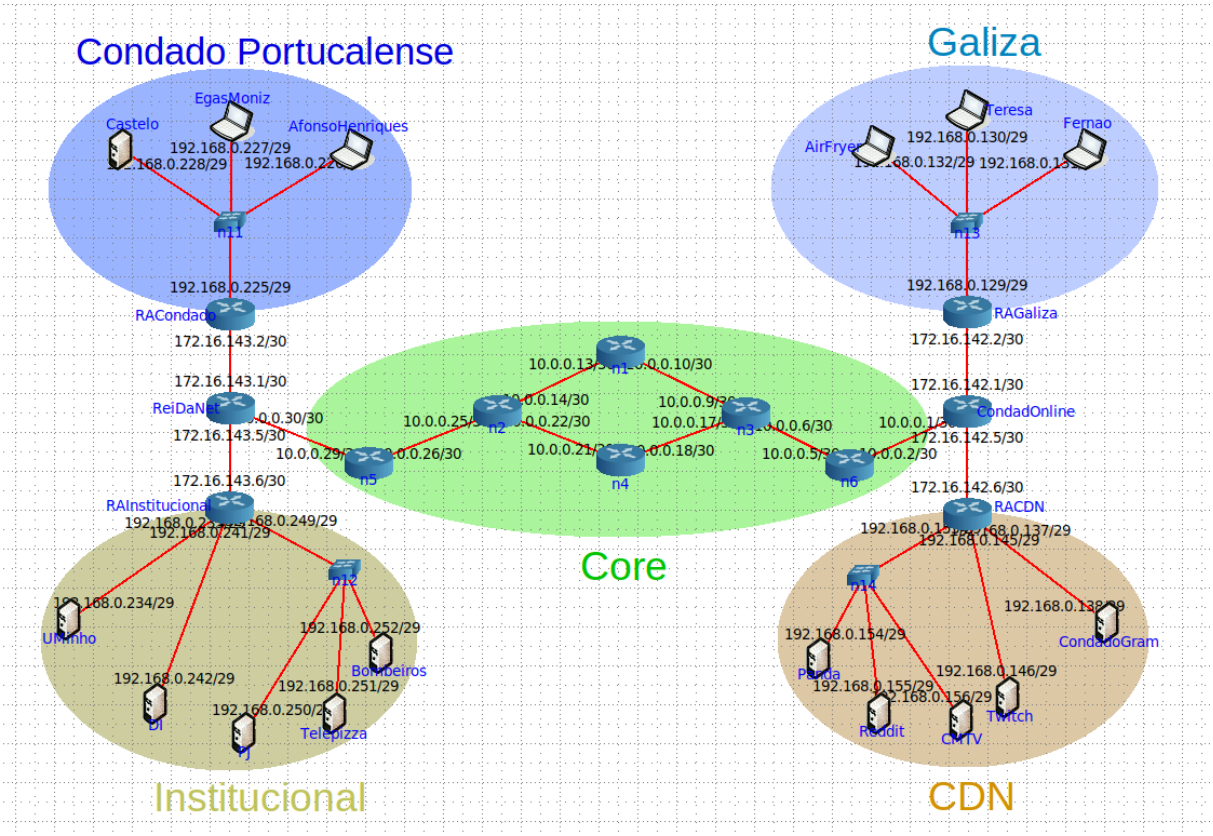



Figura 7: Topologia da Rede - 2.ª Parte.

A topologia é composta por quatro polos: Condado Portucalense, Galiza, Institucional e CDN (*Content Delivery Network*). Nos polos Condado e Galiza, existe um *router* de acesso, que permite a comunicação com o exterior, ligado a um *switch*, de modo a que todos os dispositivos partilhem a mesma rede local.

Nos polos Institucional e CDN, há três sub-redes distintas, criando uma segmentação dos dispositivos nesses polos. O Condado Portucalense e o Institucional estão ligados ao *router* do ISP ReiDaNet, enquanto Galiza e CDN estão conectados ao ISP CondadOnline. Entre os dois ISPs, há um ISP de trânsito, cuja rede *Core* é formada pelos dispositivos n1 a n6.

## 3.1 Questão 1

### 3.1.1: Problema Geral

Com os avanços da Inteligência Artificial, D. Afonso Henriques termina todas as suas tarefas mais cedo e vê-se com algum tempo livre. Decide então fazer remodelações no reino:

### 3.1.2: Questão 1.a

**Problema:** De modo a garantir uma posição estrategicamente mais vantajosa e ter casa de férias para relaxar entre batalhas, ordena a construção de um segundo Castelo, em Braga. Não tendo qualquer queixa do serviço prestado, recorre aos serviços do ISP ReiDaNet, que já utiliza no condado, para ter acesso à rede no segundo Castelo. O ISP atribuiu-lhe o endereço de rede IP 172.68.**XX**.192/26 em que **XX** corresponde ao número do nosso grupo (**PL50**). Defina um esquema de endereçamento que

permita o estabelecimento de pelo menos 6 redes e que garanta que cada uma destas possa ter 5 ou mais *hosts*. Assuma que todos os endereços de sub-redes são utilizáveis.

**Resposta:** Foi atribuído o endereço de rede 172.68.50.192/26 ao novo Castelo em Braga e é necessário definir um esquema de endereçamento que possibilite a criação de, no mínimo, 6 sub-redes, onde cada sub-rede tenha capacidade para 5 ou mais *hosts*.

**Análise Inicial:**

- **Rede /26:**
  - **Endereço:** 172.68.50.192/26;
  - **Máscara:** 255.255.255.192;
  - **Total de endereços**: 64 (intervalo de 172.68.50.192 a 172.68.50.255);
  - **Endereços utilizáveis:** 62 (descontando o endereço de rede e de *broadcast*).

**Requisito de *Hosts*:** Cada sub-rede deve ter pelo menos 5 *hosts*. Considerando que cada sub-rede possui um endereço para identificação da rede e outro para *broadcast*, é necessário dispor de um total mínimo de 7 endereços por sub-rede. Como os endereços são distribuídos em potências de 2, o menor bloco que atende a essa exigência possui 8 endereços (dos quais 6 são utilizáveis para *hosts*).

**Solução – Sub-redes /29 -** Utilizando uma máscara /29 (255.255.255.248) temos:

- Número de endereços por sub-rede: 8;
- Endereços utilizáveis: 6;
- Número de sub-redes no bloco /26:

$$64 \text{ endereços} / 8 \text{ endereços por sub-rede} = 8 \text{ sub-redes}$$

Dessa forma, temos 8 sub-redes /29, excedendo o requisito de 6 sub-redes e garantindo, em cada uma, 6 *hosts* disponíveis.

**Divisão das Sub-redes /29:**

- **172.68.50.192/29:**
  - **Intervalo:** 172.68.50.192 – 172.68.50.199; **Rede:** 172.68.50.192; ***Broadcast*:** 172.68.50.199; ***Hosts*:** 172.68.50.193 – 172.68.50.198.
- **172.68.50.200/29:**
  - **Intervalo:** 172.68.50.200 – 172.68.50.207; **Rede:** 172.68.50.200; ***Broadcast*:** 172.68.50.207; ***Hosts:*** 172.68.50.201 – 172.68.50.206.
- **172.68.50.208/29:**
  - **Intervalo:** 172.68.50.208 – 172.68.50.215; **Rede:** 172.68.50.208; ***Broadcast:*** 172.68.50.215; ***Hosts:*** 172.68.50.209 – 172.68.50.214.
- **172.68.50.216/29:**
  - **Intervalo:** 172.68.50.216 – 172.68.50.223; **Rede:** 172.68.50.216; ***Broadcast:*** 172.68.50.223; ***Hosts:*** 172.68.50.217 – 172.68.50.222.
- **172.68.50.224/29:**
  - **Intervalo:** 172.68.50.224 – 172.68.50.231; **Rede:** 172.68.50.224; ***Broadcast:*** 172.68.50.231; ***Hosts:*** 172.68.50.225 – 172.68.50.230.

- **172.68.50.232/29:**
  - **Intervalo:** 172.68.50.232 – 172.68.50.239; **Rede:** 172.68.50.232; ***Broadcast:*** 172.68.50.239; ***Hosts:*** 172.68.50.233 – 172.68.50.238.
- **172.68.50.240/29:**
  - **Intervalo:** 172.68.50.240 – 172.68.50.247; **Rede:** 172.68.50.240; ***Broadcast:*** 172.68.50.247; ***Hosts:*** 172.68.50.241 – 172.68.50.246.
- **172.68.50.248/29:**
  - **Intervalo:** 172.68.50.248 – 172.68.50.255; **Rede:** 172.68.50.248; ***Broadcast:*** 172.68.50.255; ***Hosts:*** 172.68.50.249 – 172.68.50.254.

Cada sub-rede atende ao requisito, oferecendo 6 *hosts* utilizáveis, e temos 8 sub-redes ao todo, possibilitando a alocação de redes para diferentes setores ou funções dentro do ambiente do Castelo.

**Explicação *Bitwise* da Alteração:**

- **Máscara /26 Original:** Em binário: 255.255.255.192 → Último octeto: 11000000. Com 26 *bits* fixos, há 6 *bits* disponíveis para *hosts*.
- **Máscara /29 Proposta:** Em binário: 255.255.255.248 → Último octeto: 11111000. Aqui, 29 *bits* são fixos e restam apenas 3 *bits* para *hosts*.
- **Transformação *Bitwise*:** "Empréstimo" de *Bits*: Ao mudar de /26 para /29, são "emprestados" 3 *bits* do campo de *hosts* para a identificação das sub-redes. Isto resulta no seguinte número de sub-redes: $2^3 = 8$. Cada sub-rede possui também $2^3 = 8$ endereços, dos quais 6 são utilizáveis.
- **Aumento:** O valor do último octeto incrementa de 8 em 8 (192, 200, 208, etc.), pois cada sub-rede abrange 8 endereços.

**Conclusão:** Dividir o bloco 172.68.50.192/26 em sub-redes com máscara /29 atende aos requisitos do problema, pois:

- Cria 8 sub-redes (superior ao mínimo exigido de 6).
- Cada sub-rede possui 6 *hosts* utilizáveis (suficiente para os 5 ou mais *hosts* requeridos).

Este esquema de endereçamento é adequado para garantir uma posição estrategicamente vantajosa e a infraestrutura necessária para o novo Castelo em Braga, utilizando os serviços do ISP ReiDaNet.

#### 3.1.3: Questão 1.b

**Pergunta:** Ligue um novo *host* Castelo2 diretamente ao *router* ReiDaNet. Associe-lhe um endereço, à sua escolha, pertencente a uma sub-rede disponível das criadas na alínea anterior (garanta que a interface do *router* ReiDaNet utiliza o primeiro endereço válido da sub-rede escolhida). Verifique que tem conectividade com os dispositivos do Condado Portucalense.

**Resposta:** Para integrar o novo *host* Castelo2 à rede foi efetuada uma ligação direta entre o *host* e o *router* ReiDaNet. Para isso, escolheu-se uma das sub-redes previamente criadas (172.68.50.200/29), garantindo que a interface do *router* utilize o primeiro endereço válido da sub-rede, 172.68.50.201/29, enquanto o Castelo2 foi configurado com o endereço 172.68.50.202/29. Após a configuração, foram realizados **testes de conetividade** (*ping*) aos três dispositivos do **Condado Portucalense** (endereços 192.168.0.226, 192.168.0.227 e 192.168.0.228), os quais apresentaram respostas positivas, confirmando a comunicação adequada entre os dispositivos.

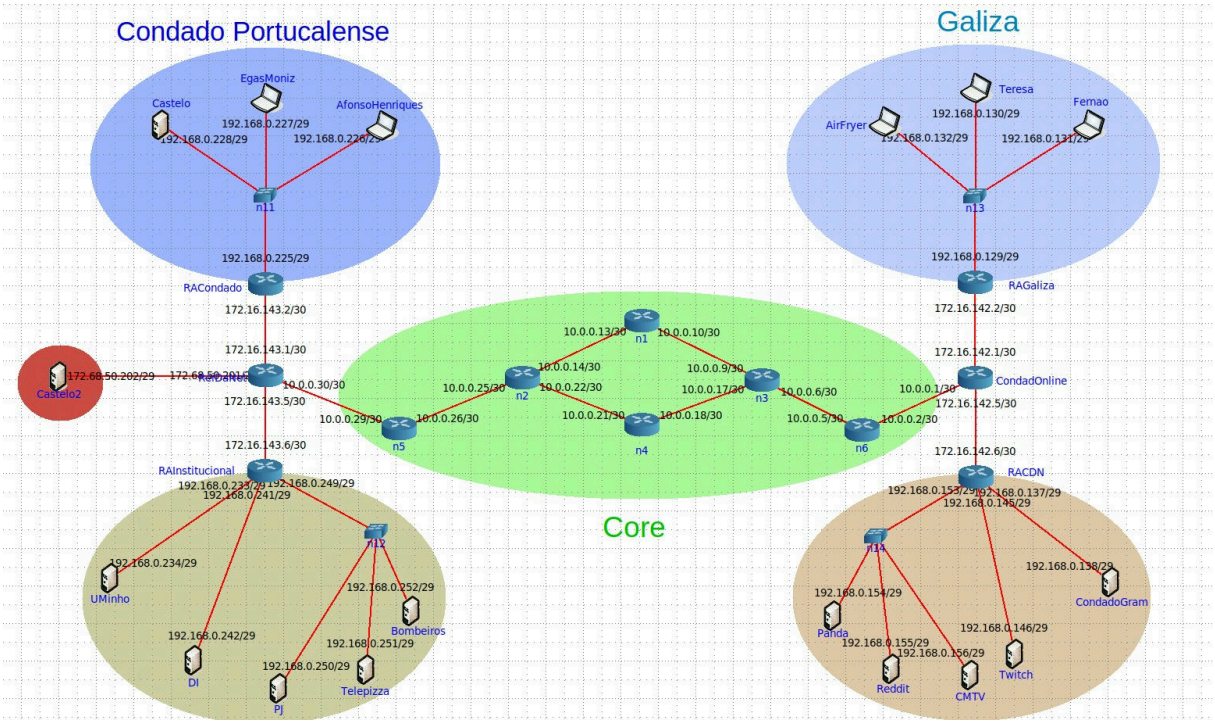



Figura 8: Topologia CORE atualizada - Ligação direta entre o Castelo2 e o *router* do ReiDaNet.

```bash
root@Castelo2:/tmp/pycore.43275/Castelo2.conf# ping 192.168.0.228
PING 192.168.0.228 (192.168.0.228) 56(84) bytes of data.
64 bytes from 192.168.0.228: icmp_seq=1 ttl=62 time 0.241 ms
64 bytes from 192.168.0.228: icmp_seq=2 ttl=62 time 0.128 ms
64 bytes from 192.168.0.228: icmp_seq=3 ttl=62 time=0.092 ms
64 bytes from 192.168.0.228: icmp_seq=4 ttl=62 time=0.092 ms
64 bytes from 192.168.0.228: icmp_seq=5 ttl=62 time=0.107 ms
64 bytes from 192.168.0.228: icmp_seq=6 ttl=62 time 0.106 ms
64 bytes from 192.168.0.228: icmp_seq=7 ttl=62 time 0.113 ms
64 bytes from 192.168.0.228: icmp_seq=8 ttl=62 time=0.107 ms
64 bytes from 192.168.0.228: icmp_seq=9 ttl=62 time=0.099 ms
64 bytes from 192.168.0.228: icmp_seq=10 ttl=62 time=0.104 ms

--- 192.168.0.228 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9212ms rtt min/avg/max/mdev = 0.092/0.118/0.241/0.041 ms
```

*Ping* do "Castelo" do Condado Portucalense a partir do Castelo2.

```bash
root@Castelo2:/tmp/pycore.43275/Castelo2.conf# ping 192.168.0.227
PING 192.168.0.227 (192.168.0.227) 56(84) bytes of data.
64 bytes from 192.168.0.227: icmp_seq=1 ttl=62 time=0.374 ms
64 bytes from 192.168.0.227: icmp_seq=2 ttl=62 time=0.156 ms
64 bytes from 192.168.0.227: icmp_seq=3 ttl=62 time=0.124 ms
64 bytes from 192.168.0.227: icmp_seq=4 ttl=62 time=0.109 ms
64 bytes from 192.168.0.227: icmp_seq=5 ttl=62 time=0.146 ms
64 bytes from 192.168.0.227: icmp_seq=6 ttl=62 time 0.119 ms
64 bytes from 192.168.0.227: icmp_seq=7 ttl=62 time=0.102 ms
64 bytes from 192.168.0.227: icmp_seq=8 ttl=62 time=0.129 ms
64 bytes from 192.168.0.227: icmp_seq=9 ttl=62 time 0.113 ms
64 bytes from 192.168.0.227: icmp_seq=10 ttl=62 time=0.127 ms

--- 192.168.0.227 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9198ms rtt min/avg/max/mdev = 0.102/0.149/0.374/0.076 ms
```

*Ping* do "EgasMoniz" do Condado Portucalense a partir do Castelo2.

```bash
root@Castelo2:/tmp/pycore.43275/Castelo2.conf# ping 192.168.0.226
PING 192.168.0.226 (192.168.0.226) 56(84) bytes of data.
64 bytes from 192.168.0.226: icmp_seq=1 ttl=62 time=0.241 ms
64 bytes from 192.168.0.226: icmp_seq=2 ttl=62 time=0.082 ms
64 bytes from 192.168.0.226: icmp_seq=3 ttl=62 time=0.107 ms
64 bytes from 192.168.0.226: icmp_seq=4 ttl=62 time 0.128 ms
64 bytes from 192.168.0.226: icmp_seq=5 ttl=62 time=0.106 ms
64 bytes from 192.168.0.226: icmp_seq=6 ttl=62 time=0.204 ms
64 bytes from 192.168.0.226: icmp_seq=7 ttl=62 time=0.326 ms
64 bytes from 192.168.0.226: icmp_seq=8 ttl=62 time=0.081 ms
64 bytes from 192.168.0.226: icmp_seq=9 ttl=62 time 0.111 ms
64 bytes from 192.168.0.226: icmp_seq=10 ttl=62 time=0.151 ms

--- 192.168.0.226 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9212ms rtt min/avg/max/mdev = 0.081/0.153/0.326/0.075 ms
```

*Ping* do "AfonsoHenriques" do Condado Portucalense a partir do Castelo2.

#### 3.1.4: Questão 1.c

**Pergunta:** Não estando satisfeito com a decoração deste novo Castelo, opta por eliminar a sua rota *default*. Adicione as rotas necessárias para que o Castelo2 continue a ter acesso ao Condado Portucalense e à rede Institucional. Mostre que a conectividade é restabelecida, assim como a tabela de encaminhamento resultante. Explicite ainda a utilidade de uma rota *default*.

**Resposta:** Neste exercício, o objetivo é restabelecer a conectividade do **Castelo2** com o **Condado Portucalense** e a **Rede Institucional**, uma vez que a rota *default* foi eliminada. Para isso, configuramos rotas estáticas específicas para cada rede, garantindo que o tráfego seja devidamente encaminhado.

**Passo 1: Remoção da Rota *Default*** - No terminal do Castelo2, removemos a rota *default* existente para forçar o uso de rotas estáticas:

```bash
ip route del default
```

**Passo 2: Configuração das Rotas Estáticas** - Tendo em atenção que:
- A rede do **Condado Portucalense** seja 192.168.0.224/29;
- A rede **Institucional** se divida em 192.168.0.232/29, 192.168.0.240/29 e 192.168.0.248/29;
- O *gateway* do **Castelo2** seja 172.68.50.201/29;
- A *interface* ativa do **Castelo2** seja `eth0`.

Adicionamos as rotas através dos seguintes comandos:

```bash
ip route add 192.168.0.224/29 via 172.68.50.201 dev eth0
ip route add 192.168.0.232/29 via 172.68.50.201 dev eth0
ip route add 192.168.0.240/29 via 172.68.50.201 dev eth0
ip route add 192.168.0.248/29 via 172.68.50.201 dev eth0
```

**Passo 3: Verificação da Conectividade** - Para confirmar que as novas rotas funcionam, utilizamos o comando *ping* para endereços nas respetivas redes, como por exemplo:

```bash
ping 192.168.0.228 # Ligação ao Castelo do Condado Portucalense
ping 192.168.0.234 # Ligação à UMinho da Rede Institucional
ping 192.168.0.242 # Ligação ao DI da Rede Institucional
ping 192.168.0.250 # Ligação à PJ da Rede Institucional
```

Como todos os *pings* realizados **obtiveram resposta**, a comunicação estava restabelecida.

**Passo 4: Exibição da Tabela de *Routing*** - Para conferir a tabela de encaminhamento e para confirmar as rotas adicionadas executamos o seguinte comando:

```bash
netstat -rn
```

O que produziu a seguinte tabela de encaminhamento:

```
root@Castelo2:/tmp/pycore.39611/Castelo2.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
172.68.50.200   0.0.0.0         255.255.255.248 U         0 0          0 eth0
192.168.0.224   172.68.50.201   255.255.255.248 UG        0 0          0 eth0
192.168.0.232   172.68.50.201   255.255.255.248 UG        0 0          0 eth0
192.168.0.240   172.68.50.201   255.255.255.248 UG        0 0          0 eth0
192.168.0.248   172.68.50.201   255.255.255.248 UG        0 0          0 eth0
```

Figura 9: Tabela de encaminhamento resultante das ações da alínea.

**Utilidade da Rota *Default*:** A rota *default* é utilizada para encaminhar pacotes destinados a redes que não tenham uma rota específica definida na tabela. Ela aponta para um *gateway* que normalmente fornece acesso à Internet ou a redes externas. No exercício, ao removermos a rota *default*, precisamos criar rotas específicas para garantir o acesso às redes necessárias. Esta abordagem garante que o **Castelo2** se comunique corretamente com o **Condado Portucalense** e a **Rede Institucional**, mesmo sem uma rota *default* configurada inicialmente.

## 3.2 Questão 2

### 3.2.1: Problema Geral

D. Afonso Henriques quer enviar fotos do novo Castelo à sua mãe, D. Teresa, mas está a ter alguns problemas de comunicação. Este alega que o problema deverá estar no dispositivo de D. Teresa, uma vez que no dia anterior conseguiu fazer *stream* de *Fortnite* para todos os seus subscritores da *Twitch*, e acabou de sair de uma discussão política no *Reddit*. O principal objetivo desta tarefa é entender o porque das dificuldades de D. Afonso Henriques e aplicar uma possível correção à luz dos princípios do **encaminhamento** de pacotes e do ***Classless InterDomain Routing*** (CIDR).

### 3.2.2: Questão 2.a

**Pergunta:** Confirme, através do comando *ping*, que **AfonsoHenriques** tem efetivamente conectividade com os servidores ***Reddit*** e ***Twitch***.

**Resposta:** Para confirmar a conectividade de **AfonsoHenriques** com os servidores do ***Reddit*** e da ***Twitch***, foram executados os seguintes comandos no terminal:

```bash
ping 192.168.0.155 # Reddit
ping 192.168.0.146 # Twitch
```

E obtiveram-se os seguintes *outputs*:

```bash
#Resultado do ping ao Reddit:
root@AfonsoHenriques:/tmp/pycore.35999/AfonsoHenriques.conf# ping 192.168.0.155
PING 192.168.0.155 (192.168.0.155) 56(84) bytes of data.
64 bytes from 192.168.0.155: icmp_seq=1 ttl=55 time=0.278 ms
64 bytes from 192.168.0.155; icmp_seq=2 ttl=55 time=0.338 ms
64 bytes from 192.168.0.155: icmp_seq=3 ttl=55 time=0.329 ms
64 bytes from 192.168.0.155; icmp_seq=1 ttl=55 time=0.356 ms
64 bytes from 192.168.0.155: icmp_seq=5 ttl=55 time=0.323 ms
64 bytes from 192.168.0.155: icmp_seq=6 ttl=55 time 0.329 ms
64 bytes from 192.168.0.155: icmp_seq=7 ttl=55 time=0.349 ms
64 bytes from 192.168.0.155: icmp_seq=8 ttl=55 time=0.341 ms
64 bytes from 192.168.0.155: icmp_seq=3 ttl=55 time=0.337 ms
64 bytes from 192.168.0.155; icmp_seq=10 ttl=55 time=0.357 ms

--- 192.168.0.155 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9225ms
rtt min/avg/max/mdev = 0.278/0.333/0.357/0.021 ms
```

```bash
#Resultado do ping à Twitch:
root@AfonsoHenriques:/tmp/pycore.35999/AfonsoHenriques.conf# ping 192.168.0.146
PING 192.168.0.146 (192.168.0.146) 56(84) bytes of data.
64 bytes from 192.168.0.146: icmp_seq=1 ttl=55 time=0.217 ms
64 bytes from 192.168.0.146: icmp_seq=2 ttl=55 time=0.182 ms
64 bytes from 192.168.0.146: icmp_seq=3 ttl=55 time=0.300 ms
64 bytes from 192.168.0.146: icmp_seq=4 ttl=55 time=0.354 ms
64 bytes from 192.168.0.146: icmp_seq=5 ttl=55 time=0.362 ms
```

```
64 bytes from 192.168.0.146: icmp_seq=6 ttl=55 time=0.339 ms
64 bytes from 192.168.0.146: icmp_seq=7 ttl=55 time=0.441 ms
64 bytes from 192.168.0.146: icmp_seq=8 ttl=55 time=0.378 ms
64 bytes from 192.168.0.146: icmp_seq=3 ttl=55 time=0.340 ms
64 bytes from 192.168.0.146: icmp_seq=10 ttl=55 time=0,267 ms

--- 192.168.0.146 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9223ms
rtt min/avg/max/mdev = 0.182/0.318/0.441/0.073 ms
```

A partir dos resultados obtidos, podemos constatar que os *pings* receberam respostas, com tempos de resposta aceitáveis, o que comprova que **AfonsoHenriques** está a conseguir estabelecer comunicação com ambos os servidores. Essa verificação evidencia que a origem do problema na transmissão das fotos não está relacionada à conectividade da máquina de **AfonsoHenriques**, mas provavelmente a um problema no dispositivo de **D.Teresa**.

Portanto, os testes demonstram que, apesar das dificuldades de comunicação relatadas por D.Teresa, D. Afonso Henriques possui acesso adequado à Internet e, em especial, aos servidores do *Reddit* e da *Twitch*.

#### 3.2.3: Questão 2.b

**Problema:** Recorrendo ao comando *netstat -rn*, analise as tabelas de encaminhamento dos dispositivos AfonsoHenriques e Teresa. Existe algum problema com as suas entradas? Identifique e descreva a utilidade de cada uma das entradas destes dois *hosts*.

**Resposta:** Inicialmente é necessário entender o significado do *output* do comando *netstat -rn*, este apresenta as seguintes categorias:

**1. *Destination*:** O endereço de destino para o qual os pacotes deverão ser enviados.

**2. *Gateway***: O endereço do *router* que deve ser usado para alcançar o destino, se aplicável.

**3. *Genmask***: A máscara de rede associada ao destino, que define a parte da sub-rede do endereço IP.

**4. *Flags***: Indica o estado da rota ou conexão (U para ativa, G para *gateway*).

**5. MSS** (*Maximum Segment Size*): O tamanho máximo do segmento de dados que pode ser enviado numa única transmissão.

**6. *Window***: O tamanho da janela de receção, que determina quantos *bytes* podem ser enviados antes de receber uma confirmação.

**7. *irtt*** (*Initial Round Trip Time*): O tempo de ida e volta inicial estimado para pacotes, usado para otimização de conexões.

**8. *Iface:*** A interface de rede associada ao rota ou conexão, indicando qual *interface* está a ser usada para a comunicação.

Alguns destes campos podem apenas apresentar o valor 0 que pode indicar que não estão definidos.

```
root@AfonsoHenriques:/tmp/pycore.33925/AfonsoHenriques.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.0.225   0.0.0.0         UG        0 0          0 eth0
192.168.0.224   0.0.0.0         255.255.255.248 U         0 0          0 eth0
```

Figura 10: Tabela de encaminhamento de D. Afonso Henriques.

```
root@Teresa:/tmp/pycore.33925/Teresa.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         192.168.0.129   0.0.0.0         UG        0 0          0 eth0
192.168.0.128   0.0.0.0         255.255.255.248 U         0 0          0 eth0
```

Figura 11: Tabela de encaminhamento de D. Teresa.

Utilizando o caso de D. Afonso Henriques, constata-se que a primeira entrada na tabela tem como destino 0.0.0.0. Este é um endereço especial que aponta para a situação em que o endereço IP do datagrama não corresponde a nenhum dos destinos específicos listados na tabela. Caso esta situação ocorra, o datagrama seria enviado para o *gateway* especificado, que neste caso é 192.168.0.225, que corresponde ao *router* RACondado. O *genmask* é definido como 0.0.0.0, porque nesta configuração não há restrições sobre os endereços de destino. A *flag* "UG" significa que a rota está ativa ("U" de *Up*) e que é uma rota para um *gateway* ("G"). A segunda entrada da tabela tem como destino 192.168.0.224 (identificador da sub-rede), com um *genmask* de 255.255.255.248, indicando que é uma sub-rede local. A *flag* "U" significa que a rota está ativo e que os pacotes destinados a esse endereço podem ser enviados diretamente, sem a necessidade de passar por um *router*. Assim, a primeira entrada lida com pacotes sem destino específico, enquanto a segunda trata de pacotes que podem ser entregues diretamente na rede local.

A análise é a mesma para o dispositivo de D. Teresa, apenas mudando o endereço do seu *gateway* e os endereços de destino. Assim sendo, pode-se afirmar que as tabelas de encaminhamento analisadas **não** estão na origem do problema de conectividade entre D. Afonso Henriques e D. Teresa.

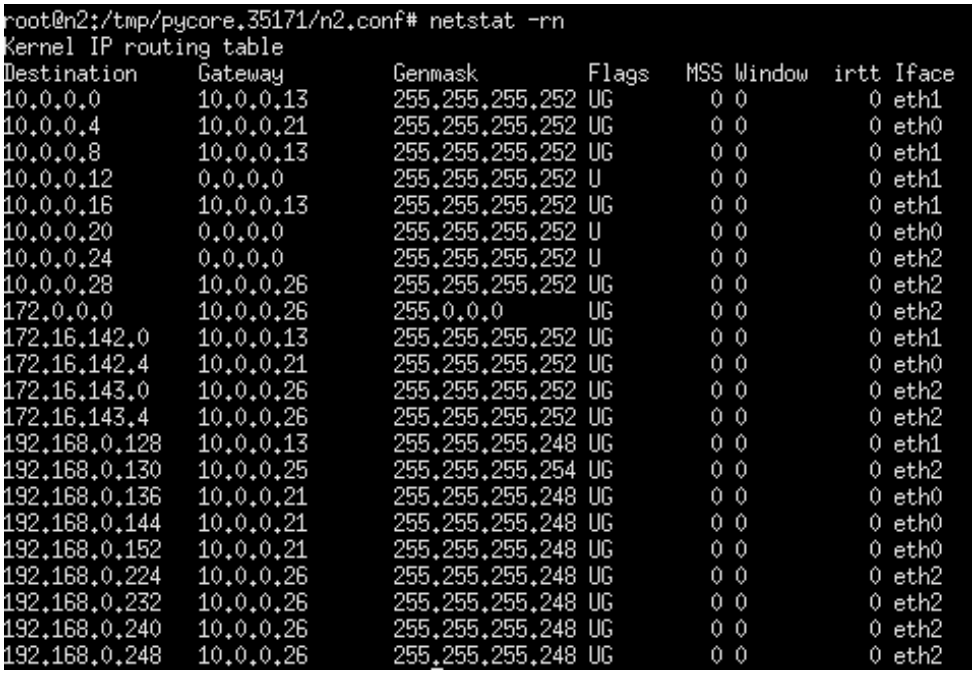
### 3.2.4: Questão 2.c

**Problema:** Analise o comportamento dos *routers* do core da rede (n1 a n6) quando tenta estabelecer comunicação entre os *hosts* AfonsoHenriques e Teresa. Indique que o(s) dispositivo(s) não permite(m) o encaminhamento correto dos pacotes. Seguidamente, avalie e explique a(s) causa(s) do funcionamento incorreto do dispositivo. Utilize o comando *ip route add/del* para adicionar as rotas necessárias ou remover rotas incorretas. Verifique a sintaxe completa do comando a usar com *man ip-route* ou *man route*. Poderá também utilizar o comando *traceroute* para se certificar do rota nó a nó. Considere a alínea resolvida assim que houver tráfego a chegar ao ISP CondadOnline.

**Resposta:** A análise começou pela utilização do comando *traceroute* no dispositivo de D.Afonso Henriques com o destino sendo o dispositivo de D.Teresa. Como se pode ver no *output* seguinte, os pacotes apenas são enviados até ao endereço 10.0.0.25, que corresponde ao *router* n2. Sendo assim, o próximo passo foi analisar a tabela de endereçamento de n2.

```bash
root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# traceroute 192.168.0.130
traceroute to 192.168.0.130 (192.168.0.130), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225)  0.026 ms  0.005 ms  0.004 ms
 2  172.16.143.1 (172.16.143.1)  0.154 ms  0.008 ms  0.005 ms
```

```
5  3 10.0.0.29 (10.0.0.29) 0.083 ms 0.012 ms 0.008 ms
6  4 10.0.0.25 (10.0.0.25) 0.045 ms 0.012 ms 0.009 ms
7  5 10.0.0.25 (10.0.0.25) 3077.360 ms H 3077,328 ms !H 3077.316 ms !H
```



```
root@n2:/tmp/pycore.35171/n2.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.0.13       255.255.255.252 UG        0 0          0 eth1
10.0.0.4        10.0.0.21       255.255.255.252 UG        0 0          0 eth0
10.0.0.8        10.0.0.13       255.255.255.252 UG        0 0          0 eth1
10.0.0.12       0.0.0.0         255.255.255.252 U         0 0          0 eth1
10.0.0.16       10.0.0.13       255.255.255.252 UG        0 0          0 eth1
10.0.0.20       0.0.0.0         255.255.255.252 U         0 0          0 eth0
10.0.0.24       0.0.0.0         255.255.255.252 U         0 0          0 eth2
10.0.0.28       10.0.0.26       255.255.255.252 UG        0 0          0 eth2
172.0.0.0       10.0.0.26       255.0.0.0       UG        0 0          0 eth2
172.16.142.0    10.0.0.13       255.255.255.252 UG        0 0          0 eth1
172.16.142.4    10.0.0.21       255.255.255.252 UG        0 0          0 eth0
172.16.143.0    10.0.0.26       255.255.255.252 UG        0 0          0 eth2
172.16.143.4    10.0.0.26       255.255.255.252 UG        0 0          0 eth2
192.168.0.128   10.0.0.13       255.255.255.248 UG        0 0          0 eth1
192.168.0.130   10.0.0.25       255.255.255.254 UG        0 0          0 eth2
192.168.0.136   10.0.0.21       255.255.255.248 UG        0 0          0 eth0
192.168.0.144   10.0.0.21       255.255.255.248 UG        0 0          0 eth0
192.168.0.152   10.0.0.21       255.255.255.248 UG        0 0          0 eth0
192.168.0.224   10.0.0.26       255.255.255.248 UG        0 0          0 eth2
192.168.0.232   10.0.0.26       255.255.255.248 UG        0 0          0 eth2
192.168.0.240   10.0.0.26       255.255.255.248 UG        0 0          0 eth2
192.168.0.248   10.0.0.26       255.255.255.248 UG        0 0          0 eth2
```

Figura 12: Tabela de endereçamento de n2.

Ao explorar a tabela, detetamos a existência de uma rota direta para o dispositivo de D. Teresa, 192.168.0.130. No entanto, esta regra possuía uma máscara 255.255.255.254, ou em notação CIDR /31, o que é impossível, uma vez que neste caso não restam endereços para *hosts*. Assim sendo, utilizamos o comando abaixo para remover a rota incorreta.

```bash
route del -net 192.168.0.130 netmask 255.255.255.254
```

Deste modo tentamos executar o comando *traceroute* mais uma vez:

```bash
root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# traceroute 192.168.0.130
traceroute to 192.168.0.130 (192.168.0.130), 30 hops max, 60 byte packets
 1 192.168.0.225 (192.168.0.225) 0.076 ms 0.006 ms 0.004 ms
 2 172.16.143.1 (172.16.143.1) 0.015 ms 0.006 ms 0.005 ms
 3 10.0.0.29 (10.0.0.29) 0.017 ms 0.007 ms 0.006 ms
 4 10.0.0.25 (10.0.0.25) 0.026 ms 0.009 ms 0.009 ms
 5 10.0.0.13 (10.0.0.13) 0.090 ms 0.014 ms 0.011 ms
 6 10.0.0.25 (10.0.0.25) 0.011 ms 0.018 ms 0.012 ms
 7 10.0.0.13 (10.0.0.13) 0.012 ms 0.011 ms 0.012 ms
 8 * * *
 9 * * *
10 * * *
11 * * *
12 * * *
13 * 10.0.0.13 (10.0.0.13) 0.060 ms 0.016 ms
```

```
16  14 10.0.0.25 (10.0.0.25) 0.017 ms 0.016 ms 0.016 ms
17  15 10.0.0.13 (10.0.0.13) 0.017 ms 0.017 ms 0.017 ms
18  16 10.0.0.25 (10.0.0.25) 0.016 ms 0.017 ms *
19  * * *
20  * * *
```

O que produziu um resultado no mínimo curioso: um ciclo no qual os pacotes percorrem a rota n1 -> n2 -> n1 -> n2 infinitamente.

É possível concluir, mesmo antes de ver a tabela de endereçamento de n1, que esta possui alguma regra errada, na qual tenta redirecionar o pacote com o endereço IP que iria para D. Teresa para o *router* anterior.

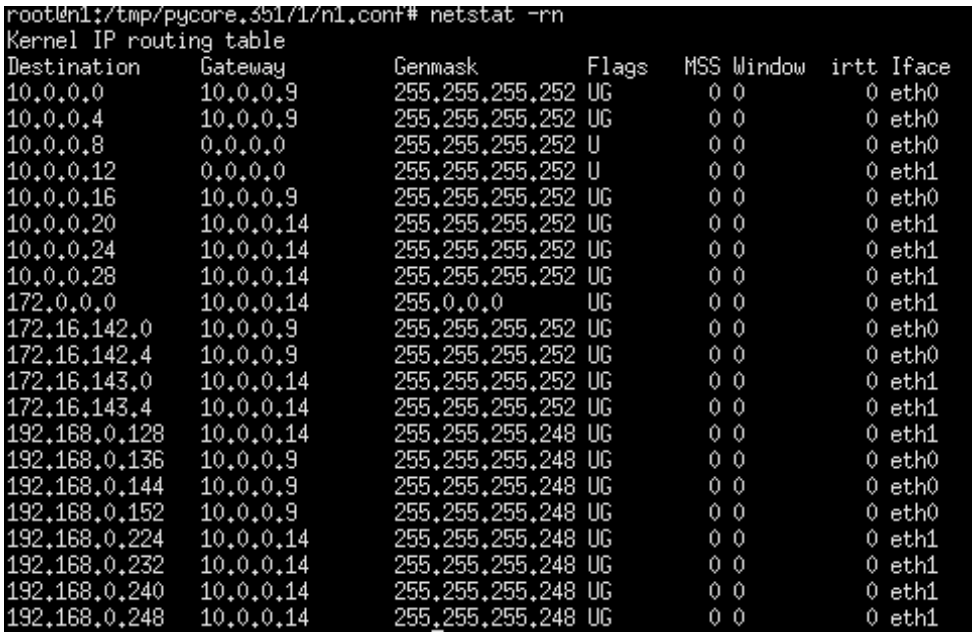



Figura 13: Tabela de endereçamento de n1.

E aqui está, o endereço 192.168.0.128 identificador da sub-net onde se encontra o dispositivo de D. Teresa, tem como *Gateway* o *router* n2 que por sua vez encaminha para n1. Para resolver apenas é necessário mudar o *Gateway* para n3, o que pode ser feito através dos seguintes comandos:

```bash
route del -net 192.168.0.128 netmask 255.255.255.248
route add -net 192.168.0.128 netmask 255.255.255.248 gw 10.0.0.9
```

Novamente *traceroute*:

```
root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# traceroute 192.168.0.130
traceroute to 192.168.0.130 (192.168.0.130), 30 hops max, 60 byte packets
 1  192.168.0.225 (192.168.0.225) 0.079 ms 0.007 ms 0.004 ms
 2  172.16.143.1 (172.16.143.1) 0.015 ms 0.006 ms 0.010 ms
 3  10.0.0.29 (10.0.0.29) 0.043 ms 0.009 ms 0.007 ms
 4  10.0.0.25 (10.0.0.25) 0.021 ms 0.009 ms 0.010 ms
 5  10.0.0.13 (10.0.0.13) 0.050 ms 0.013 ms 0.011 ms
 6  10.0.0.17 (10.0.0.17) 0.175 ms !N 0.022 ms !N *
```

Os pacotes não passam de n3, como tal é necessário identificar o problema:

```
root@n3:/tmp/pycore.35171/n3.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        10.0.0.5        255.255.255.252 UG        0 0          0 eth2
10.0.0.4        0.0.0.0         255.255.255.252 U         0 0          0 eth2
10.0.0.8        0.0.0.0         255.255.255.252 U         0 0          0 eth0
10.0.0.12       10.0.0.10       255.255.255.252 UG        0 0          0 eth0
10.0.0.16       0.0.0.0         255.255.255.252 U         0 0          0 eth1
10.0.0.20       10.0.0.18       255.255.255.252 UG        0 0          0 eth1
10.0.0.24       10.0.0.18       255.255.255.252 UG        0 0          0 eth1
10.0.0.28       10.0.0.10       255.255.255.252 UG        0 0          0 eth0
172.0.0.0       10.0.0.10       255.0.0.0       UG        0 0          0 eth0
172.16.142.0    10.0.0.5        255.255.255.252 UG        0 0          0 eth2
172.16.142.4    10.0.0.5        255.255.255.252 UG        0 0          0 eth2
172.16.143.0    10.0.0.18       255.255.255.252 UG        0 0          0 eth1
172.16.143.4    10.0.0.10       255.255.255.252 UG        0 0          0 eth0
192.168.0.136   10.0.0.5        255.255.255.248 UG        0 0          0 eth2
192.168.0.144   10.0.0.5        255.255.255.248 UG        0 0          0 eth2
192.168.0.152   10.0.0.5        255.255.255.248 UG        0 0          0 eth2
192.168.0.224   10.0.0.18       255.255.255.248 UG        0 0          0 eth1
192.168.0.232   10.0.0.10       255.255.255.248 UG        0 0          0 eth0
192.168.0.240   10.0.0.10       255.255.255.248 UG        0 0          0 eth0
192.168.0.248   10.0.0.10       255.255.255.248 UG        0 0          0 eth0
```

Figura 14: Tabela de endereçamento de n3.

Constata-se que falta a regra para encaminhar pacotes para a sub-rede 192.168.0.128, o que pode ser feito com o seguinte comando:

```bash
route add -net 192.168.0.128 netmask 255.255.255.248 gw 10.0.0.5
```

Pela última vez comando *traceroute*:

```bash
root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# traceroute 192.168.0.130
traceroute to 192.168.0.130 (192.168.0.130), 30 hops max, 60 byte packets
 1  192.168.0.255 (192.168.0.225) 0.028 ms 0.005 ms 0.004 ms
 2  172.16.143.1 (172.16.143.1) 0.015 ms 0.006 ms 0.006 ms
 3  10.0.0.29 (10.0.0.29) 0.017 ms 0.007 ms 0.007 ms
 4  10.0.0.25 (10.0.0.25) 0.018 ms 0.010 ms 0.010 ms
 5  10.0.0.13 (10.0.0.13) 0.019 ms 0.011 ms 0.011 ms
 6  10.0.0.17 (10.0.0.17) 0.033 ms 0.021 ms 0.014 ms
 7  10.0.0.5 (10.0.0.5) 0.021 ms 0.014 ms 0.015 ms
 8  10.0.0.1 (10.0.0.1) 0.022 ms 0.016 ms 0.016 ms
```

Sendo 10.0.0.1 o IP de uma das *interfaces* do *router* CondadOnline, podemos dar por concluída esta fase. No entanto, ainda resta a pergunta:

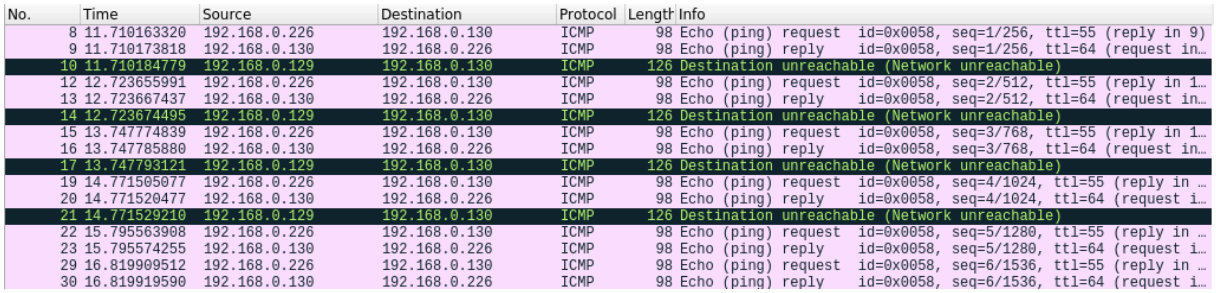
Porque D.Afonso Henriques não consegue comunicar com D.Teresa?

### 3.2.5: Questão 2.d

**Problema:** Uma vez que o *core* da rede esteja a encaminhar corretamente os pacotes enviados por D. Afonso Henriques, confira com o *Wireshark* se estes são recebidos por Teresa.

- **Alínea I**: Em caso afirmativo, porque é que continua a não existir conectividade entre D.Teresa e D.Afonso Henriques? Efetue as alterações necessárias para garantir que a conectividade é restabelecida e o confronto entre os dois é evitado.

**Resposta:** Para o efeito segue a captura do *Wireshark* na entrada do dispositivo de D. Teresa, aquando o comando *ping* por parte de D. Afonso:

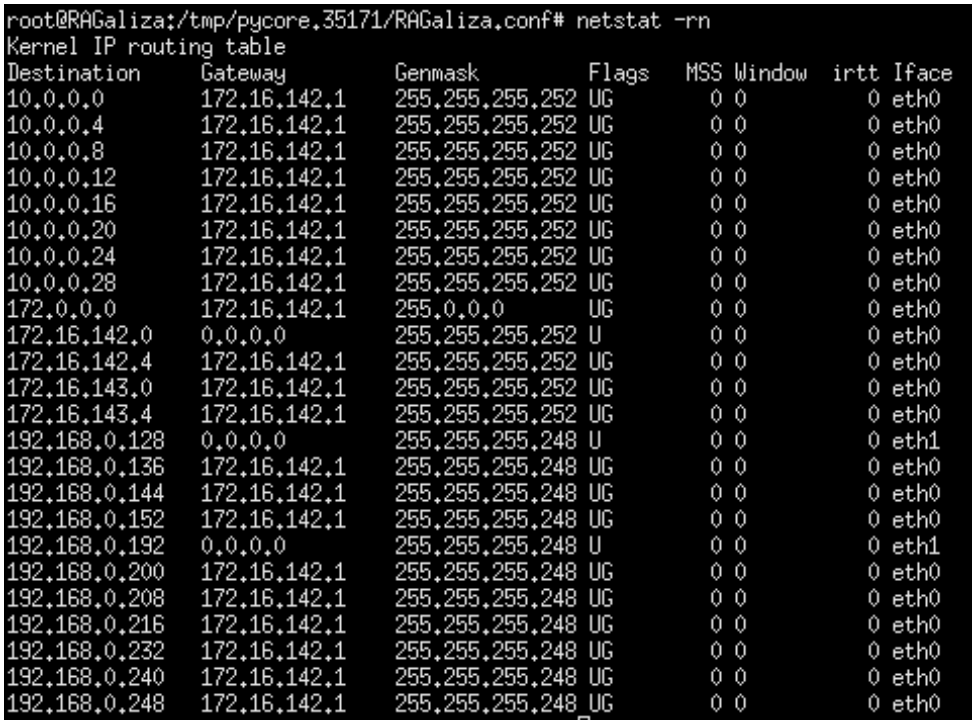


| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 8 | 11.710163320 | 192.168.0.226 | 192.168.0.130 | ICMP | 98 | Echo (ping) request id=0x0058, seq=1/256, ttl=55 (reply in 9) |
| 9 | 11.710173818 | 192.168.0.130 | 192.168.0.226 | ICMP | 98 | Echo (ping) reply id=0x0058, seq=1/256, ttl=64 (request in… |
| 10 | 11.710184779 | 192.168.0.129 | 192.168.0.130 | ICMP | 126 | Destination unreachable (Network unreachable) |
| 12 | 12.723655991 | 192.168.0.226 | 192.168.0.130 | ICMP | 98 | Echo (ping) request id=0x0058, seq=2/512, ttl=55 (reply in 1… |
| 13 | 12.723667437 | 192.168.0.130 | 192.168.0.226 | ICMP | 98 | Echo (ping) reply id=0x0058, seq=2/512, ttl=64 (request in… |
| 14 | 12.723674495 | 192.168.0.129 | 192.168.0.130 | ICMP | 126 | Destination unreachable (Network unreachable) |
| 15 | 13.747774839 | 192.168.0.226 | 192.168.0.130 | ICMP | 98 | Echo (ping) request id=0x0058, seq=3/768, ttl=55 (reply in 1… |
| 16 | 13.747785880 | 192.168.0.130 | 192.168.0.226 | ICMP | 98 | Echo (ping) reply id=0x0058, seq=3/768, ttl=64 (request in… |
| 17 | 13.747793121 | 192.168.0.129 | 192.168.0.130 | ICMP | 126 | Destination unreachable (Network unreachable) |
| 19 | 14.771505077 | 192.168.0.226 | 192.168.0.130 | ICMP | 98 | Echo (ping) request id=0x0058, seq=4/1024, ttl=55 (reply in … |
| 20 | 14.771520477 | 192.168.0.130 | 192.168.0.226 | ICMP | 98 | Echo (ping) reply id=0x0058, seq=4/1024, ttl=64 (request i… |
| 21 | 14.771529210 | 192.168.0.129 | 192.168.0.130 | ICMP | 126 | Destination unreachable (Network unreachable) |
| 22 | 15.795563908 | 192.168.0.226 | 192.168.0.130 | ICMP | 98 | Echo (ping) request id=0x0058, seq=5/1280, ttl=55 (reply in … |
| 23 | 15.795574255 | 192.168.0.130 | 192.168.0.226 | ICMP | 98 | Echo (ping) reply id=0x0058, seq=5/1280, ttl=64 (request i… |
| 29 | 16.819909512 | 192.168.0.226 | 192.168.0.130 | ICMP | 98 | Echo (ping) request id=0x0058, seq=6/1536, ttl=55 (reply in … |
| 30 | 16.819919590 | 192.168.0.130 | 192.168.0.226 | ICMP | 98 | Echo (ping) reply id=0x0058, seq=6/1536, ttl=64 (request i… |

Figura 15: Captura *Wireshark* - D.Teresa.

É evidente que os pacotes chegam até ao dispositivo de D. Teresa, no entanto a confirmação da chegada não retorna para o dispositivo de D. Afonso Henriques como pode ser visto pela mensagem "*Destination Unreachable* (*Network Unreachable*)".

Após alguma pesquisa é possível entender que a origem do problema situa-se na tabela de endereçamento do *router* RAGaliza, na qual não existe qualquer rota para a sub-net 192.168.0.224, onde se encontra o dispositivo de D. Afonso.



```
root@RAGaliza:/tmp/pycore.35171/RAGaliza.conf# netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
10.0.0.0        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.4        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.8        172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.12       172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.16       172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.20       172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.24       172.16.142.1    255.255.255.252 UG        0 0          0 eth0
10.0.0.28       172.16.142.1    255.255.255.252 UG        0 0          0 eth0
172.0.0.0       172.16.142.1    255.0.0.0       UG        0 0          0 eth0
172.16.142.0    0.0.0.0         255.255.255.252 U         0 0          0 eth0
172.16.142.4    172.16.142.1    255.255.255.252 UG        0 0          0 eth0
172.16.143.0    172.16.142.1    255.255.255.252 UG        0 0          0 eth0
172.16.143.4    172.16.142.1    255.255.255.252 UG        0 0          0 eth0
192.168.0.128   0.0.0.0         255.255.255.248 U         0 0          0 eth1
192.168.0.136   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.144   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.152   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.192   0.0.0.0         255.255.255.248 U         0 0          0 eth1
192.168.0.200   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.208   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.216   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.232   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.240   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
192.168.0.248   172.16.142.1    255.255.255.248 UG        0 0          0 eth0
```

Figura 16: Tabela de endereçamento de RAGaliza.

De modo a restaurar a conexão pode ser usado o comando:

```bash
route add -net 192.168.0.224 netmask 255.255.255.248 gw 172.16.142.1
```

E como pode ser visto pelo uso do comando *ping*, a conexão entre D.Afonso Henriques e D.Teresa foi restabelecida.

```bash
root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# ping 192.168.0.130 -c 5
PING 192.168.0.130 (192.168.0.130) 56(84) bytes of data.
```

```
3   64 bytes from 192.168.0.130: icmp_seq=1 ttl=55 time=0.094 ms
4   64 bytes from 192.168.0.130: icmp_seq=2 ttl=55 time=0.104 ms
5   64 bytes from 192.168.0.130: icmp_seq=3 ttl=55 time=0.105 ms
6   64 bytes from 192.168.0.130: icmp_seq=1 ttl=55 time=0.102 ms
7   64 bytes from 192.168.0.130: icmp_seq=5 ttl=55 time=0.105 ms
8   --- 192.168.0.130 ping statistics ---
9   5 packets transmitted, 5 received, 0% packet loss, time 4095ms
10  rtt min/avg/max/mdev = 0.094/0.102/0.105/0.004 ms
```

- **Alínea II**: As rotas dos pacotes ICMP *echo reply* são as mesmas, mas em sentido inverso, que as rotas dos pacotes ICMP *echo request* enviados entre AfonsoHenriques e Teresa? (Sugestão: analise as rotas nos dois sentidos com o *traceroute*). Mostre graficamente a rota seguida nos dois sentidos por esses pacotes ICMP.

**Resposta:** Utilizaram-se os seguintes *outputs* do *traceroute*, primeiramente de D. Afonso para D. Teresa, e de seguida, D. Teresa para D. Afonso para criar o seguinte grafo orientado:

```
1   root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# traceroute 192.168.0.130
2   traceroute to 192.168.0.130 (192.168.0.130), 30 hops max, 60 byte packets
3   1 192.168.0.225 (192.168.0.225) 0.288 ms 0.009 ms 0.008 ms
4   2 172.16.143.1 (172.16.143.1) 0.150 ms 0.009 ms 0.009 ms
5   3 10.0.0.29 (10.0.0.29) 0.131 ms 0.009 ms 0.008 ms
6   4 10.0.0.25 (10.0.0.25) 0.022 ms 0.014 ms 0.009 ms
7   5 10.0.0.13 (10.0.0.13) 0.143 ms 0.016 ms 0.012 ms
8   6 10.0.0.17 (10.0.0.17) 0.166 ms 0.062 ms 0.014 ms
9   7 10.0.0.5 (10.0.0.5) 0.229 ms 0.018 ms 0.015 ms
10  8 10.0.0.1 (10.0.0.1) 0.169 ms 0.022 ms 0.057 ms
11  9 172.16.142.2 (172.16.142.2) 0.057 ms 0.024 ms 0.033 ms
12  10 192.168.0.130 (192.168.0.130) 0.077 ms 0.023 ms 0.026 ms
```
bash

```
1   root@Teresa:/tmp/pycore.35171/Teresa.conf# traceroute 192.168.0.226
2   traceroute to 192.168.0.226 (192.168.0.226), 30 hops max, 60 byte packets
3   1 192.168.0.129 (192.168.0.129) 0.158 ms 0.006 ms 0.004 ms
4   2 172.16.142.1 (172.16.142.1) 0.043 ms 0.008 ms 0.006 ms
5   3 10.0.0.2 (10.0.0.2) 0.083 ms 0.009 ms 0.015 ms
6   4 10.0.0.6 (10.0.0.6) 0.071 ms 0.010 ms 0.009 ms
7   5 10.0.0.18 (10.0.0.18) 0.097 ms 0.012 ms 0.010 ms
8   6 10.0.0.14 (10.0.0.14) 0.033 ms 0.031 ms 0.013 ms
9   7 10.0.0.26 (10.0.0.26) 0.129 ms 0.018 ms 0.014 ms
10  8 10.0.0.30 (10.0.0.30) 0.052 ms 0.018 ms 0.016 ms
11  9 172.16.143.2 (172.16.143.2) 0.091 ms 0.021 ms 0.019 ms
12  10 192.168.0.226 (192.168.0.226) 0.058 ms 0.023 ms 0.020 ms
```
bash

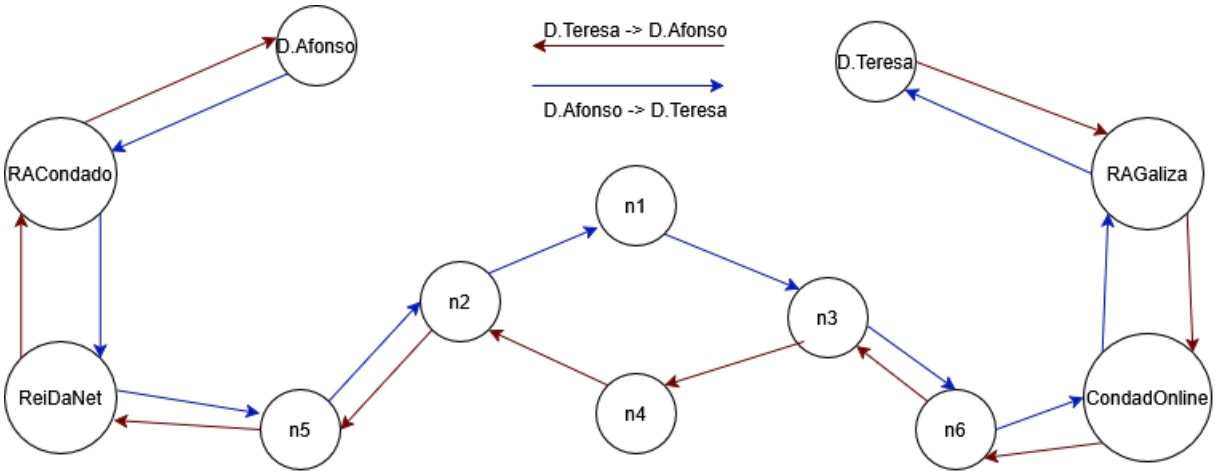



Figura 17: Grafo das rotas dos pacotes entre dispostivos D.Teresa e D.Afonso.

Por análise do grafo, pode-se concluir que as rotas dos pacotes *echo reply* são diferentes das rotas dos pacotes *echo requests*. Isto deve-se às configurações de cada tabela de encaminhamento dentro da topologia.

### 3.2.6: Questão 2.e

**Problema:** Estando restabelecida a conectividade entre os dois *hosts*, obtenha a tabela de encaminhamento de n5 e foque-se na seguinte entrada:

```
ip route 192.168.0.0 255.255.255.0 10.0.0.30
```

Figura 18: Entrada tabela encaminho n5.

Existe uma correspondência (*match*) nesta entrada para pacotes enviados para o polo Galiza? E para CDN? Caso seja essa a entrada utilizada para o encaminhamento, permitirá o funcionamento esperado do dispositivo? Ofereça uma explicação pela qual essa entrada é ou não utilizada.

**Resposta:** Para que haja correspondência, a entrada na tabela de encaminhamento de n5 deve incluir uma rota para as redes que correspondem ao polo da Galiza e a CDN. Neste caso deverá pelos menos alcançar o *router* CondadOnline (10.0.0.1) que conecta tanto Galiza como CDN.

Sendo o endereço IP 192.168.0.0 e a máscara 255.255.255.0 ou seja /24 seria possível identificar todos os *hosts* entre 192.168.0.1 a 192.168.0.254, o que efetivamente permitiria criar a rota para todos os dispositivos de Galiza e CDN. No entanto, há um problema, o *Gateway* é 10.0.0.30, o que corresponde exatamente ao *router* no sentido oposto de Galiza e CDN. Assim sendo, apenas com esta entrada na tabela de encaminhamento os pacotes não poderiam ser enviados pelo *router* para Galiza e CDN, mas sim para o Condado Portucalense e Institucional. Logo para o efeito esperado seria necessário mudar o *Gateway* da entrada para 10.0.0.25.

Pode ficar a pergunta: Como os pacotes chegam ao destino com esta configuração? A resposta é que esta entrada não é usada devido a outras entradas na tabela de encaminhamento com um maior prefixo (*longest prefix match*). O *longest prefix match* faz com que o *router* escolha a rota com o prefixo mais longo, ou seja, a mais específica. Isto garante que, mesmo com a entrada na tabela errada, as rotas mais específicas, desde que bem definidas, sejam utilizadas para encaminhar os pacotes ao destino correto.

As quatro entradas abaixo permitem a chegada dos pacotes ao polo Galiza e à CDN, mais especificamente às sub-redes dentro dos mesmos. A primeira refere-se ao polo Galiza, enquanto as restantes pertencem ao polo CDN. Todas têm uma máscara de 29 *bits*, obviamente superior aos 24 *bits* apresen-

tados na entrada de referência. Por isso, o tráfego funciona corretamente, mesmo com a existência da entrada original.

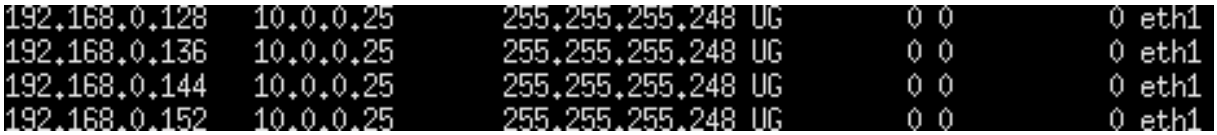



Figura 19: Entradas tabela encaminho n5 para Galiza e CDN.

#### 3.2.7: Questão 2.f

**Problema:** Os endereços utilizados pelos quatro polos são endereços públicos ou privados? E os utilizados no *core* da rede/ISPs? Justifique convenientemente.

**Resposta:** Os endereços utilizados pelos 4 polos são endereços privados, uma vez que a máscara CIDR é /16, ou seja, o endereço começa por 192.168.x.x, e, como foi lecionado nas aulas teóricas, se o endereço estiver dentro desse intervalo, é um endereço privado. O mesmo acontece para os endereços usados no *core*, que começam por 10.0.x.x, ou seja, a máscara CIDR é /8. Assim, conclui-se que todos os endereços nos 4 polos, mais os do *core*, são privados.

#### 3.2.8: Questão 2.g

**Problema:** Os *switches* localizados em cada um dos polos têm um endereço IP atribuído? Porquê?

**Resposta:** Os *switches* não têm nenhum endereço IP associado, pois, como a sua função principal é transmitir dados, ou *frames*, não precisam de ter um endereço IP. Algo que também pode justificar esta ausência é o facto de que os *switches* usam os endereços MAC *(Media Access Control)* ao invés dos endereços IP.

### 3.3 Questão 3

#### 3.3.1: Problema Geral

Ao ver as fotos no *CondadoGram*, D. Teresa não ficou convencida com as novas alterações e ordena que D. Afonso Henriques vá arrumar o castelo. Inconformado, este decide planear um novo ataque, mas constata que o seu exército não só perde bastante tempo a decidir que direção tomar a cada salto como, por vezes, inclusivamente se perde.

#### 3.3.2: Questão 3.a

**Problema:** De modo a facilitar a travessia, elimine as rotas referentes a Galiza e CDN no dispositivo n6 e defina um esquema de sumarização de rotas (*Supernetting*) que permita o uso de apenas uma rota para ambos os polos. Confirme que a conectividade é mantida.

**Resposta:** Primeiramente foi necessário identificar as travessias referentes a Galiza e CDN no dispositivo n6, para evitar problemas com o *longest prefix match*. Assim sendo, após análise das rotas de n6 com o comando *netstat -rn* , removemos as seguintes rotas:

```bash
route del -net 172.16.142.0 netmask 255.255.255.252
route del -net 172.16.142.4 netmask 255.255.255.252
route del -net 192.168.0.128 netmask 255.255.255.248
```

```
route del -net 192.168.0.136 netmask 255.255.255.248
route del -net 192.168.0.144 netmask 255.255.255.248
route del -net 192.168.0.152 netmask 255.255.255.248
```

Após este procedimento, começamos a realizar o processo de sumarização no qual agrupamos o IP 192.168.0.128/29 com o IP 192.168.0.136/29 em 192.168.0.128/28, e o IP 192.168.0.144/29 com o IP 192.168.0.152/29 em 192.168.0.144/28. Finalmente agruparam-se estes dois novos endereços em 192.168.0.128/27. Quanto aos endereços entre os *routers* agrupou-se 172.16.142.0/30 com 172.16.142.4/30 resultando em 172.16.142.0/28. Com os seguintes comandos, é possível adicionar as novas rotas sumarizadas:

```bash
route add -net 192.168.0.128 netmask 255.255.255.224 gw 10.0.0.1
route add -net 172.16.142.0 netmask 255.255.255.240 gw 10.0.0.1
```

Para testar a conexão utilizou-se o comando *ping* de D. Afonso para D. Teresa e para CMTV, para verificar a conexão de CondadoPortucalense com Galiza e CDN, respetivamente. Como pode ser visto comprova-se a correção da nova configuração.

```bash
root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# ping 192.168.0.130 -c 5
PING 192.168.0.130 (192.168.0.130) 56(84) bytes of data.
64 bytes from 192.168.0.130: icmp_seq=1 ttl=55 time=0.141 ms
64 bytes from 192.168.0.130: icmp_seq=2 ttl=55 time 0.143 ms
64 bytes from 192.168.0.130: icmp_seq=3 ttl=55 time=0.334 ms
64 bytes from 192.168.0.130: icmp_seq=1 ttl=55 time 0.141 ms
64 bytes from 192.168.0.130: icmp_seq=5 ttl=55 time 0.330 ms
--- 192.168.0.130 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4097ms
rtt min/avg/max/mdev = 0.141/0.217/0.334/0.093 ms
```

```bash
root@AfonsoHenriques:/tmp/pycore.35171/AfonsoHenriques.conf# ping 192.168.0.156 -c 5
PING 192.168.0.156 (192.168.0.156) 56(84) bytes of data.
64 bytes from 192.168.0.156: icmp_seq=1 ttl=55 time 0.171 ms
64 bytes from 192.168.0.156: icmp_seq=2 ttl=55 time 0.128 ms
64 bytes from 192.168.0.156: icmp_seq=3 ttl=55 time=0.128 ms
64 bytes from 192.168.0.156: icmp_seq=1 ttl=55 time 0.128 ms
64 bytes from 192.168.0.156: icmp_seq=5 ttl=55 time 0.129 ms
--- 192.168.0.156 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4089ms
rtt min/avg/max/mdev = 0.128/0.136/0.171/0.017 ms
```

#### 3.3.3: Questão 3.b

**Problema:** Repita o processo descrito na alínea anterior para CondadoPortucalense e Institucional, também no dispositivo n6.

**Resposta:** Da mesma forma da alínea *a*, foi necessário identificar as rotas a sumarizar, com o auxílio do comando *netstat -rn* identificamos 6 rotas que foram eliminadas com os seguintes comandos:

```bash
route del -net 172.16.143.0 netmask 255.255.255.252
route del -net 172.16.143.4 netmask 255.255.255.252
route del -net 192.168.0.224 netmask 255.255.255.248
route del -net 192.168.0.232 netmask 255.255.255.248
route del -net 192.168.0.240 netmask 255.255.255.248
route del -net 192.168.0.248 netmask 255.255.255.248
```

Agrupamos o IP 192.168.0.224/29 com o IP 192.168.0.232/29 em 192.168.0.128/28, e o IP 192.168.0.240/29 com o IP 192.168.0.248/29 em 192.168.0.144/28. Finalmente agruparam-se estes dois novos endereços em 192.168.0.224/27. Quanto aos endereços entre os *routers* agruparam-se 172.16.143.0/30 com 172.16.143.4/30 resultando em 172.16.143.0/28. Com os seguintes comandos, é possível adicionar as novas rotas sumarizadas:

```bash
route add -net 192.168.0.224 netmask 255.255.255.224 gw 10.0.0.6
route add -net 172.16.143.0 netmask 255.255.255.240 gw 10.0.0.6
```

Para testar a conexão utilizou-se o comando *ping* de D. Teresa para D. Afonso e para UMinho, para verificar a conexão de Galiza com CondadoPortucalense e Institucional, respetivamente. Como pode ser visto comprova-se a correção da nova configuração:

```bash
root@Teresa:/tmp/pycore.41991/Teresa.conf# ping 192.168.0.226 -c 5
PING 192.168.0.226 (192.168.0.226) 56(84) bytes of data.
64 bytes from 192.168.0.226: icmp_seq=1 ttl=55 time 0.184 ms
64 bytes from 192.168.0.226: icmp_seq=2 ttl=55 time 0.132 ms
64 bytes from 192.168.0.226: icmp_seq=3 ttl=55 time=0.132 ms
64 bytes from 192.168.0.226: icmp_seq=4 ttl=55 time=0.130 ms
64 bytes from 192.168.0.226: icmp_seq=5 ttl=55 time 0.129 ms
--- 192.168.0.226 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4069ms
rtt min/avg/max/mdev = 0.129/0.141/0.184/0.021 ms
```

```bash
root@Teresa:/tmp/pycore.41991/Teresa.conf# ping 192.168.0.234 -c 5
PING 192,168,0,234 (192,168,0,234) 56(84) bytes of data,
64 bytes from 192.168.0.234: icmp_seq=1 ttl=55 time=0,175 ms
64 bytes from 192.168.0.234: icmp_seq=2 ttl=55 time=0,332 ms
64 bytes from 192.168.0.234: icmp_seq=3 ttl=55 time=0,128 ms
64 bytes from 192.168.0.234: icmp_seq=4 ttl=55 time=0,328 ms
64 bytes from 192.168.0.234: icmp_seq=5 ttl=55 time=0,162 ms
—-- 192.168.0.234 ping statistics —--
5 packets transmitted, 5 received, 0% packet loss, time 4075ms
rtt min/avg/max/mdev = 0.128/0.225/0.332/0.087 ms
```

#### 3.3.4: Questão 3.c

**Problema:** Comente os aspetos positivos e negativos do uso do *Supernetting*.

**Resposta:** *Supernetting* é uma técnica que combina várias pequenas redes numa única rede maior, simplificando a tabela de *routing*. Isto reduz o número de rotas, tornando o processo de *routing* mais eficiente e económico em termos de processamento. Além disso, também permite um melhor uso dos

endereços IP, evitando desperdícios e facilitando a administração da rede. Com menos rotas para gerir, a manutenção da rede torna-se mais simples.

Contudo, *Supernetting* também tem as suas desvantagens. Ao agrupar várias sub-redes, perde-se flexibilidade, o que pode dificultar a aplicação de políticas específicas ou a segmentação de tráfego. Além disso, a técnica pode ser difícil de aplicar em redes com endereços não contíguos, levando à fragmentação. Isso também pode dificultar o isolamento de tráfego e a aplicação de medidas de segurança. Em redes grandes, a gestão de super-redes pode ser mais complexa, e alguns protocolos de *routing* podem ter dificuldades para lidar com elas, gerando problemas de convergência.

Em síntese, *Supernetting* é útil para simplificar o processo de *routing*, mas deve ser usado com cautela, considerando a flexibilidade e as necessidades de segurança da rede.

# 4. Conclusão

O Trabalho Prático N.º 2 sobre o **Protocolo IPv4** proporcionou-nos uma compreensão prática das técnicas de endereçamento e encaminhamento IP, como ***CIDR***, ***Subnetting*** e **sumarização de rotas (*Supernetting*)**. Através da utilização da máquina virtual, mais propriamente, do simulador de redes ***CORE***, com finalidade de simular uma rede real, foi possível aplicar conceitos teóricos e resolver problemas de conectividade, ao utilizar comandos como *ping* e *traceroute* para diagnosticar erros nas tabelas de endereçamento dos dispositivos.

Este projeto destacou a importância de uma configuração adequada das rotas e a necessidade de soluções a longo prazo, como a transição para o **IPv6**, para enfrentar a exaustão de endereços. A implementação do conceito de ***Supernetting*** demonstrou como simplificar a gestão de redes, de forma a reduzir a complexidade das tabelas de encaminhamento. De forma geral, este trabalho permitiu **consolidar conhecimentos teóricos** e **desenvolver habilidades práticas essenciais** para uma melhor compreensão das **Redes de Computadores**.

# 5. Bibliografia


[1] K. W. Kurose J. F. & Ross, *Computer Networking: A Top-Down Approach*, 8.º ed. Pearson, 2020.

[2] G. Kessler, «RFC 2151 - A Primer On Internet and TCP/IP Tools and Utilities». [Online]. Disponível em: https://datatracker.ietf.org/doc/html/rfc2151

[3] U. o. S. C. Information Sciences Institute, «RFC 709 - Internet Protocol». [Online]. Disponível em: https://datatracker.ietf.org/doc/html/rfc791

[4] J. Postel, «RFC 792 - Internet Control Message Protocol». [Online]. Disponível em: https://datatracker.ietf.org/doc/html/rfc792

[5] Wireshark Foundation, «Wireshark - Documentation». [Online]. Disponível em: https://wireshark.org/docs/