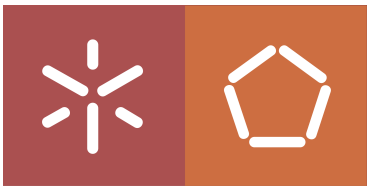

Universidade do Minho
Escola de Engenharia

Engenharia de Serviços em Rede

Streaming de áudio e vídeo a pedido e em tempo real

Relátorio Trábalho Prático 2

Grupo **PL43**

MEI - 1º Ano - 1º Semestre

Trabalho realizado por:

PG53601 - Afonso Xavier Cardoso Marques
PG55093 - Pedro Alexandre da Silva Oliveira
PG55097 - Vasco Rafael Barroso Gonçalves Rito

Braga, 4 de dezembro de 2024

Índice

Lista de Figuras

1. Introdução

Neste trabalho prático, o desafio proposto consiste no desenvolvimento de um serviço Over the Top (OTT) para a entrega de conteúdos multimédia, com foco na transmissão de vídeo em tempo real. O objetivo principal é prototipar uma solução que utilize uma rede de overlay composta por Points of Presence (POPs), permitindo a entrega eficiente e escalável de vídeos aos utilizadores finais. Este tipo de arquitetura visa não apenas melhorar a qualidade da transmissão, mas também otimizar a utilização da rede, oferecendo maior robustez, flexibilidade e capacidade de adaptação a diferentes condições de tráfego e procura.

A topologia escolhida para testar este trabalho é composta por dois servidores, um bootstrapper e outro não-bootstrapper, três Points of Presence (POPs), cinco nós intermédios e cinco clientes.

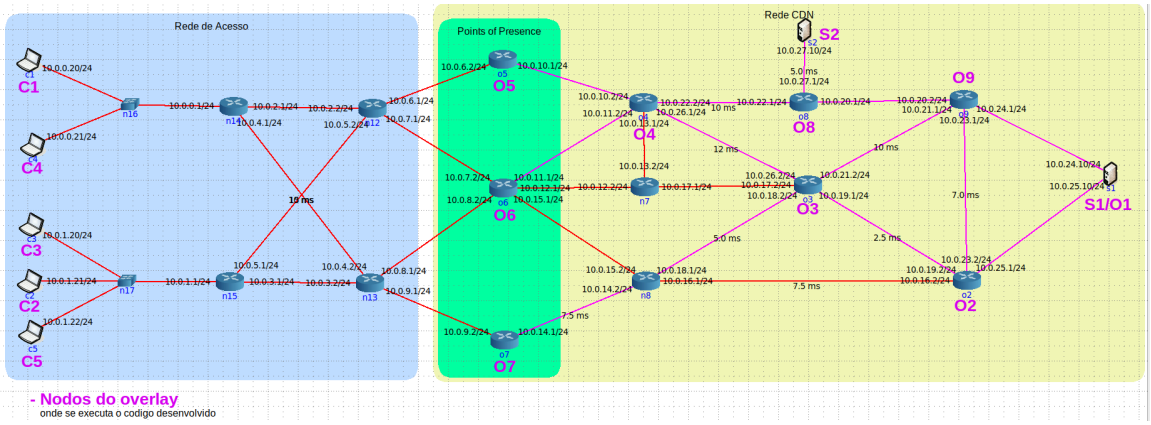

Figura 1: Topologia escolhida

2. Arquitetura da Solução

Neste trabalho é pretendido que se crie um sistema de entrega de conteúdo multimédia em tempo real partindo de um servidor de *streaming* para um conjunto de clientes. Para isso os nodos Points of Presence, ao qual nos iremos referir como POPs, irão estar na fronteira da rede de distribuição de conteúdo, *Content Distribution Network* ou CDN, e da rede de acesso, fazendo de ponto de contacto dos clientes com a CDN. Os POPs recebem o conteúdo via *multicast* e depois enviam-no via *unicast* para cada um dos clientes que pretendem aceder àquele conteúdo. A seguinte imagem mostra a arquitetura geral de um sistema com o mesmo fim.

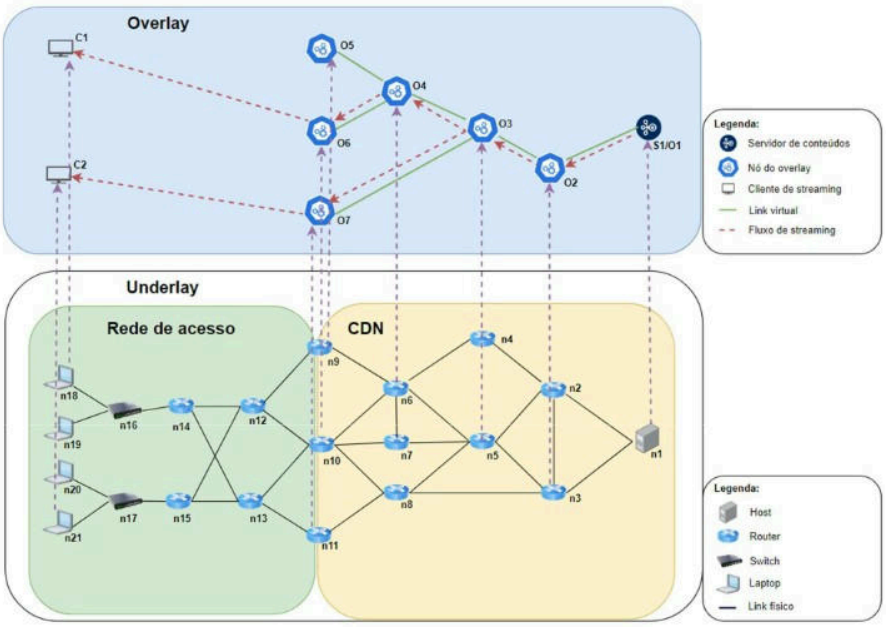

Figura 2: Arquitetura do Sistema

Para a implementação do trabalho prático utilizamos a linguagem de programação Python pois, além de possuir bibliotecas que facilitam o uso de *sockets* e *threads*, é também a linguagem com que o grupo se sente mais à vontade. Para a construção da rede de *overlay* decidimos utilizar um ficheiro de configuração em formato JSON, onde está presente a informação que cada nodo irá utilizar. Segue em baixo a explicação dos argumentos necessários e a estrutura dos ficheiros de configuração.

2.1. Script

De forma a tornar mais rápida e simples a execução da topologia decidimos desenvolver um pequeno *script* que será aplicado apenas nos nodos que atuam no serviço de *streaming*, isto é, os servidores, nodos intermédios, POPs e os clientes.

```
#!/bin/bash

# Comandos iniciais para configurar o ambiente

cd /home/core/Desktop/ESR-2425/TP2/code

boot_ip="10.0.25.10"
topologia="topologias/topo_overlay_complex.json"

# Verifique o nome do nó para decidir o comando específico
case $1 in
  "c")
    export DISPLAY=:0.0
    python3 oClient.py "$boot_ip" "$2"
    ;;
  "n")
    python3 oNode.py -n "$boot_ip"
    ;;
  "pop")
    python3 oNode.py -pop "$boot_ip"
    ;;
  "s")
    python3 oNode.py -s "$topologia" "$2"
    ;;
  *)
    echo "Node not recognized"
    ;;
esac
```

Figura 3: Script para correr Topologia

2.2. Servidores de Streaming

Os servidores de *streaming* são responsáveis por enviar o conteúdo pela rede CDN até aos POPs. Estes servidores, de forma a serem inicializados, devem saber qual é o ficheiro JSON que contem a configuração da topologia e uma *flag* que define se o servidor será ou não o *bootstrapper*. O *bootstrapper* é o único nó que coordena a descoberta da rede e deve ser sempre iniciado em primeiro lugar. Em todas as topologias, deve apenas existir um único servidor *bootstrapper*. O seguinte comando permite iniciar o servidor:

```
python3 oNode.py -s "topologia.json" 1
```

- A *flag* '-s' indica que o nó é do tipo servidor.
- O número '1', no final, especifica que o servidor será inicializado como *bootstrapper*.
- Se for '0', o servidor será inicializado como não *bootstrapper*, ficando apenas disponível para distribuir conteúdo.

2.3. Points of Present (POP)

Os POPs são responsáveis por retransmitir o conteúdo recebido pelos servidores para os clientes. Atuam na fronteira entre as duas redes, sendo os únicos nodos que mantêm ligação continua com o cliente durante a transmissão do video. A sua implementação permite escalabilidade e eficiência na transmissão, sendo assim capazes de lidar com vários clientes ligados ao mesmo tempo.

Para serem inicializados, os POPs necessitam de saber o endereço IP do servidor que atua como *bootstrapper*. Falando com o servidor neste endereço, os POPs conseguem descobrir e conectar-se aos seus vizinhos na topologia. Para inicializar um POP, utiliza-se o comando:

```
python3 oNode.py -pop "ip_bootstrapper"
```

- A *flag* '-pop' indica que o nodo é um POP.
- "ip_bootstrapper" deve ser substituído pelo endereço IP do *bootstrapper*.

2.4. Nodo intermédio

Este tipo de nodo apenas trata de receber e encaminhar os pacotes relativos às diversas *streams* desde o servidor até aos POPs. Possui uma base de dados onde guarda todos os seus vizinhos, quem vai receber pacotes de uma determinada *stream* e toda a lógica de escolha de caminhos para construir o fluxo de transmissão. De forma a ser inicializado, apenas é necessário indicar a *flag* '-n' e o endereço do *bootstrapper*.

```
python3 oNode.py -n "ip_bootstrapper"
```

2.5. Cliente

Os clientes são os nodos finais da rede que solicitam e consomem o conteúdo transmitido. Cada cliente conecta-se à rede falando primeiro com o servidor que atua como *bootstrapper* de forma a conseguir descobrir todos os servidores e POPs responsáveis por fornecer o conteúdo solicitado. Antes de executar o cliente, é necessário configurar a variável de ambiente "DISPLAY", que define o ecrã onde será exibido o conteúdo. Para inicializar um cliente, utilizam-se os seguintes comandos:

```
export DISPLAY=:0.0
python3 oClient.py "ip_bootstrapper" "movie_name"
```

- "ip_bootstrapper" deve ser substituído pelo IP do servidor *bootstrapper*.
- "movie_name" representa o nome do conteúdo que será transmitido.

3. Especificação dos protocolos

Com o objetivo de transmitir conteúdos de streaming, foi utilizado o RTP (Realtime Transport Protocol), um protocolo de transporte especificamente desenvolvido para streaming e a entrega de conteúdo em tempo real. No seu funcionamento, o RTP utiliza o UDP como protocolo da camada de transporte, o que permite certa tolerância à perda de pacotes. Contudo, é sensível a atrasos na entrega, o que pode afetar a sequência de disposição dos pacotes, e não garante a qualidade de serviço (QoS).

O protocolo RTP é normalmente utilizado em conjunto com o RTSP (Real-Time Streaming Protocol). Enquanto o RTP é responsável pelo transporte dos dados, o RTSP é utilizado para controlar a transmissão de multimédia.

3.1. Formato das mensagens protocolares

As quatro componentes comunicam entre si através de mensagens protocolares com conteúdo especifico ao proposito que se pretende. Algumas mensagens possuem estruturas complexas que permitem atualizar as métricas para as rotas enquanto que outras apenas existem para iniciar um processo ou enviar uma porta para comunicação. Em baixo, segue a especificação das mensagens trocadas entre as componentes:

POPs e nodos intermédios

- REQ_NEIGHBOURS - mensagem utilizada para solicitar a lista de vizinhos ao *bootstrapper*;
- STATUS - mensagens protocolares que são propagadas pela rede com origem nos servidores. Reencaminha para os seus vizinhos;
- NAK - mensagem usada quando o vizinho não consegue encontrar a *stream* solicitada.

Cliente

- LOGIN - o cliente indica ao *bootstrapper* que quer ver uma *stream* e recebe dele uma lista de POPs onde se pode conectar. Esta mensagem contem apenas a porta onde o cliente vai receber a lista dos POPs;
- CONNECT_POP - o cliente, após escolher o POP, envia para ele a porta por onde vai decorrer a comunicação em RTSP;
- SEND_RTSP - mensagens em RTSP enviadas ao POP para pedir *streaming* de um vídeo, pausar e terminar.

Servidores

- RESPOND_LOGIN - a resposta do servidor *bootstrapper* ao cliente consiste em enviar a lista de todos os POPs;
- STATUS - mensagem de status que todos os servidores enviam aos nodos do *overlay*. Possui uma estrutura com os campos "servername", "time", "jumps" e "visited";
- DATA - mensagens responsáveis pelo envio de dados. Estas mensagens contêm os dados relativos aos pacotes RTP, que são reencaminhados pelos nós intermédios até chegarem aos POPs, que os enviam para os clientes;
- SEND_NEIGHBOURS - mensagem utilizada para enviar a lista de vizinhos.

4. Implementação

4.1. Etapa 1 - Construção da Topologia Overlay e Underlay

Para a construção da nossa topologia de *overlay* iremos utilizar a abordagem 2 que foi sugerida no enunciado, por isso, foi estabelecido que na topologia deve existir pelo menos um servidor que atua como *bootstrapper*, ao qual será fornecido um ficheiro de configuração JSON. Após a leitura do ficheiro, aguarda até que todos os nodos lá declarados estejam em pé e entrem em contacto com ele.

Cada novo nó *overlay* inicializado vai-se registar através do envio de uma mensagem, num *socket* TCP, ao *bootstrapper*, de forma a passar a ter conhecimento de quem são os seus vizinhos. No que diz respeito à situação de saída de um dos nós, o servidor irá informar que algum dos nós falhou.

Assim que todos os nodos estão conectados, os serviço está pronto para começar a aceitar pedidos de *streams* de clientes. Entretanto, todos os servidores iniciam um processo paralelo onde enviam, por *flooding* controlado, mensagens de monitorização pela rede *overlay*, na porta 4444, com informação sobre o servidor de origem, o instante temporal em que foi enviada, o nº de saltos que a mensagem dá e os nodos por onde já passou, de forma a evitar repetições em ciclo. Estas mensagens, ao serem recebidas pelos nodos da rede *overlay*, permitem-lhes atualizar as métricas necessárias para posteriormente ser possível construir os fluxos de *stream*. Estas mensagens são enviadas a cada 30 segundos, sendo este o intervalo de tempo que se considerou razoável para o efeito.

O ficheiro JSON segue a seguinte estrutura, onde a lista de "neighbours" corresponde apenas aos endereços IP que estão dentro da rede *overlay* e a lista "names" contém todos os endereços, em *overlay* e *underlay*, pelo qual um nodo é conhecido. Na lista "neighbours", endereços de servidores são identificados com um sufixo "s". Os nodos com prefixo "o" possuem uma lista "pop" com os endereços onde os clientes se podem conectar. Se esta lista estiver vazia, o nodo não atua como POP.

```
{
    "s1": {
        "neighbours": ["10.0.25.1", "10.0.24.1"],
        "names": ["10.0.25.10", "10.0.24.10"]
    },
    "s2": {
        "neighbours": ["10.0.27.1"],
        "names": ["10.0.27.10"]
    },
    "o2": {
        "neighbours": ["10.0.19.1", "10.0.25.10s", "10.0.23.1"],
        "names": ["10.0.25.1", "10.0.19.2", "10.0.23.2", "10.0.16.2"],
        "pop": []
    },
    "o3": {
        "neighbours": ["10.0.26.1","10.0.19.2","10.0.14.1", "10.0.21.1"],
        "names": ["10.0.26.2", "10.0.18.2", "10.0.19.1", "10.0.17.2", "10.0.21.2"],
        "pop": []
    },
    "o4": {
        "neighbours": ["10.0.26.2","10.0.11.1","10.0.10.1", "10.0.22.1"],
        "names": ["10.0.10.2", "10.0.11.2", "10.0.26.1", "10.0.13.1", "10.0.22.2"],
        "pop": []
    },
    "o5": {
        "neighbours": ["10.0.10.2"],
        "names": ["10.0.10.1", "10.0.6.2"],
        "pop": ["10.0.6.2"]
    },
    "o6": {
        "neighbours": ["10.0.11.2"],
        "names": ["10.0.7.2", "10.0.8.2", "10.0.15.1", "10.0.12.1", "10.0.11.1"],
        "pop": ["10.0.7.2", "10.0.8.2"]
    },
    "o7": {
        "neighbours": ["10.0.18.2"],
        "names": ["10.0.9.2", "10.0.14.1"],
        "pop": ["10.0.9.2"]
    },
    "o8": {
        "neighbours": ["10.0.22.2", "10.0.20.2", "10.0.27.10s"],
        "names": ["10.0.22.1", "10.0.27.1", "10.0.20.1"],
        "pop": []
    },
    "o9": {
        "neighbours": ["10.0.20.1", "10.0.21.2", "10.0.23.2", "10.0.24.10s"],
        "names": ["10.0.20.2", "10.0.21.1", "10.0.23.1", "10.0.24.1"],
        "pop": []
    }
}
```


Figura 4: JSON da topologia

4.2. Etapa 2 - Construção oClient

Na aplicação que permite que um cliente receba os conteúdos da CDN, **oClient**, o cliente primeiro autentica-se falando com o servidor *bootstrapper*, recebendo como resposta uma lista com todos os nodos POP. Como critério de escolha, o cliente liga-se ao POP que apresentar menor latência ao executar o comando *ping*.

Escolhido o POP, o cliente executa o pedido de *stream*, indicando qual o ficheiro que pretende visualizar. Surgindo a interface gráfica, o cliente deve primeiro realizar a configuração da rota, clicando para isso no botão "SETUP", que dá ordem ao POP para estabelecer na topologia *overlay* o fluxo necessário para trazer os pacotes RTP desde o servidor até ele. É também criada uma nova *stream* para aquele cliente, com um número de sessão único e a porta RTP especificada.

Outros comandos na interface do cliente incluem o "PLAY", o "PAUSE" e o "TEARDOWN". O "PLAY" dá ordem ao POP para criar um *socket* para que o cliente possa receber os pacotes da *stream*.

O "PAUSE" simplesmente coloca a *stream* em pausa e o "TEARDOWN" dá ordem para parar de receber pacotes da *stream* e fechar a interface gráfica do cliente.

4.3. Etapa 3 - oNode e Serviço de Streaming

A aplicação **oNode** pode ser executada em nodos que atuem como servidores, intermédios e POPs. Quando é executada em servidores *bootstrapper*, inicia o processo de envio dos vizinhos a cada nodo que se vai registando, envia mensagens de monitorização para que os nodos mantenham a informação sobre o estado da rede atualizada e lidam também com os registos dos clientes.

Quando executado num nó intermédio, este recebe os seus vizinhos e fica à escuta na porta 4444 para receber as mensagens de monitorização enviadas pelos servidores ativos. Estes nodos possuem uma base de dados, *database.py*, onde é guardada a seguinte informação:

- ***neighbours*:** lista com todos os vizinhos do nodo
- ***serversNeighbours*:** lista com todos os vizinhos que sejam servidor
- ***serverStatus*:** dicionário que monitoriza o estado de servidores na rede
- ***streamsDict*:** dicionário que armazena informações sobre as *streams* que atravessam o nodo.
- ***routeStreamDict*:** dicionário que guarda as rotas para as *streams*. Para cada *stream*, armazena as métricas dos nodos que a possuam.

Similarmente, os POPs também executam esta aplicação, realizando as mesmas ações dos nodos intermédios, com a particularidade destes nodos aceitarem pedidos de *stream* por parte dos clientes. Quando um cliente pede *stream* de um ficheiro, o nodo POP cria uma instância própria de *ServerWorker* para lidar com todos os pedidos RTSP que o cliente faça e para lhe fazer chegar os pacotes RTP da *stream*.

4.4. Etapa 4 - Construção Árvores de Distribuição

A construção dos fluxos de *stream* dependem da base de dados que existe em cada nodo intermédio e POP. Ao receber um pedido de *stream*, isto é o cliente clicou em "SETUP", o POP vai tentar encontrar uma *stream* na rede. Para isso, primeiro deverá consultar a lista de *streams* que estão a decorrer em si próprio. Se já tiver a fazer *stream* desse ficheiro, apenas adiciona o novo cliente à lista de receptores dessa *stream*. Caso a *stream* desejada não esteja lá, deve então ir procurá-la a um dos servidores ou a um vizinho. Para decidir onde vai procurar, verifica se já existe alguma rota na rede para a *stream* desejada. Caso esse valor seja nulo, deverá perguntar recursivamente até chegar a um servidor, recorrendo à função *getBestMetricsServerStatus* da *database*. Aqui, através das métricas recebidas nas mensagens do tipo STATUS, consegue obter o melhor vizinho com quem pode falar para fazer chegar a sua mensagem ao servidor. Esse vizinho, que recebe a mensagem na porta 6666, repete o processo, até que a mensagem chegue ao servidor.

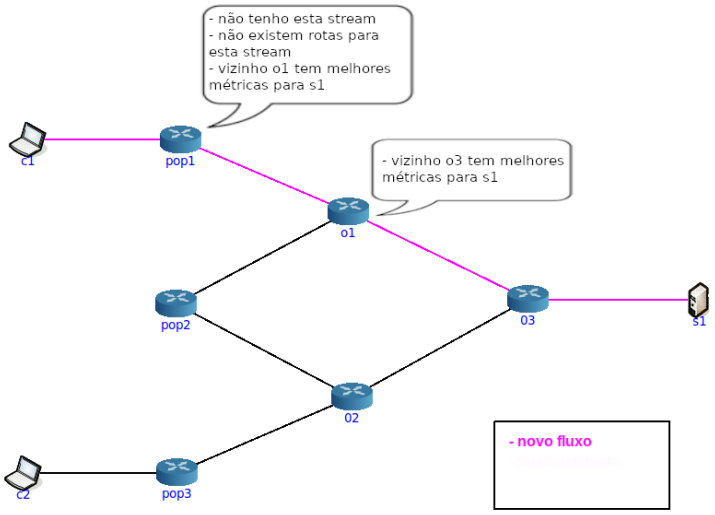

Figura 5: Pedir recursivamente ao servidor

Caso já exista uma rota para a *stream* pedida, o POP usa a função *getBestMetricsRouteStreamDict* do *database*, que consulta o dicionário *routeStreamDict* para obter o vizinho mais próximo com a *stream* desejada.

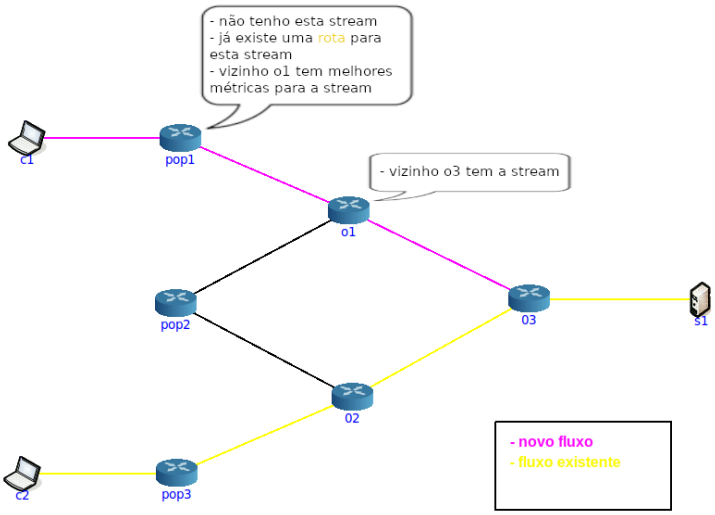

Figura 6: Pedir recursivamente até atingir um nodo com a stream

As duas funções mencionadas acima são analisam as métricas de forma hierárquica, isto é, primeiro realizam uma comparação entre as *timestamps*, de forma a saber qual dos vizinhos contactados tem menor tempo de resposta, e apenas nos casos onde os tempos de respostas sejam muito próximos, ou seja a variação temporal é inferior a 10%, se vai olhar para o número de saltos. A razão para esta decisão deve-se ao facto do *timestamp* ser uma métrica mais completa, no sentido de ser mais preciso e de oferecer uma visão temporal detalhada de eventos, enquanto o número de saltos se limita a medir a quantidade de passos intermediários entre dois pontos, sem fornecer informações sobre o tempo ou o contexto completo da comunicação.

5. Testes e resultados

5.1. Perguntas e Respostas

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et aut officiis debitis aut rerum necessitatibus saepe eveniet, ut et voluptates repudiandae sint et molestiae non recusandae. Itaque earum rerum defuturum, quas natura.

6. Conclusões e trabalho futuro

Com a conclusão deste trabalho prático, reconhece-se a sua relevância para o aprofundamento dos conhecimentos sobre o funcionamento de protocolos de *streaming* de vídeo, a comunicação entre diferentes clientes e servidores, bem como a transmissão de dados através do RTP.

Destaca-se como aspeto positivo o cumprimento integral dos requisitos do serviço, alcançando os principais objetivos definidos pela equipa docente e, ainda, a primeira etapa complementar que visa fazer a monitorização da Rede Overlay.

No futuro, poderiam ser desenvolvidas funcionalidades adicionais, como a implementação da segunda etapa complementar que tem como objetivo fornecer um método de recuperação de falhas.