



**UMinho**

**Mestrado Engenharia  
Informática**

Engenharia de Serviços em Redes  
(2023/24)

**Serviço Over the Top para  
entrega de multimédia**

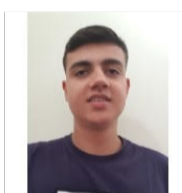
Maria S. Pires Ferreira R. Lima

25 de junho de 2024

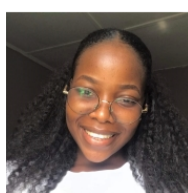
## Elementos do Grupo e Contribuição



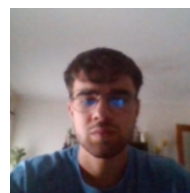
A95323



PG53944



PG51636



P54247

Cont.	Número de Aluno	Nome
X	A95323	Henrique Ribeiro Fernandes
X	PG53944	João Pedro Moreira Brito
X	PG51636	Nsimba Teresa Armando
X	PG54247	Telmo Leonel Silva Oliveira

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Arquitetura da Solução</b>	<b>2</b>
2.1	Classes Constituintes . . . . .	2
2.1.1	RtpPacket . . . . .	2
2.1.2	VideoStream . . . . .	2
2.1.3	Client . . . . .	2
2.1.4	Server . . . . .	3
2.1.5	Nos . . . . .	3
2.2	Rp . . . . .	3
<b>3</b>	<b>Especificação dos Protocolos</b>	<b>3</b>
3.1	Formatos das Mensagens Protocolares . . . . .	4
3.2	Interações . . . . .	4
<b>4</b>	<b>Implementação</b>	<b>5</b>
4.1	Construção da Topologia Overlay . . . . .	5
4.2	Serviço de Streaming . . . . .	5
4.3	Monitorização dos Servidores de conteúdos . . . . .	6
4.4	Construção dos Fluxos para Entrega de Dados . . . . .	6
<b>5</b>	<b>Limitações da Solução</b>	<b>7</b>
<b>6</b>	<b>Testes e Resultados</b>	<b>7</b>
<b>7</b>	<b>Conclusão e Trabalhos Futuros</b>	<b>9</b>
	<b>Bibliografia</b>	<b>10</b>

# 1 Introdução

Nos últimos 50 anos, a Internet passou por uma transformação notável, abandonando a comunicação sistema-a-sistema para abraçar um consumo contínuo e em tempo real de diversos conteúdos. Essa mudança desafiadora demanda uma reavaliação da infraestrutura IP, originalmente não projetada para essa intensidade. A solução atual recai sobre redes de entrega de conteúdo (CDNs) e serviços Over the Top (OTT), desenvolvidos na camada aplicacional.

Este projeto visa conceber e prototipar um serviço OTT, focando na otimização de recursos para melhorar a experiência do usuário. Utilizando o emulador CORE como base, o objetivo é criar um protótipo de entrega de áudio/vídeo/texto em tempo real, partindo de um servidor de conteúdo para N clientes.

O protótipo incluirá a designação de um Rendezvous Point (RP) para receber conteúdo em unicast, que será distribuído por uma rede de nós intermediários, formando uma árvore de distribuição otimizada para entregar conteúdo de maneira eficiente, com o mínimo de atraso e largura de banda necessária. A **Figura 1** destaca a arquitetura geral, evidenciando a importância da estrutura do overlay aplicacional na entrega eficiente de conteúdo

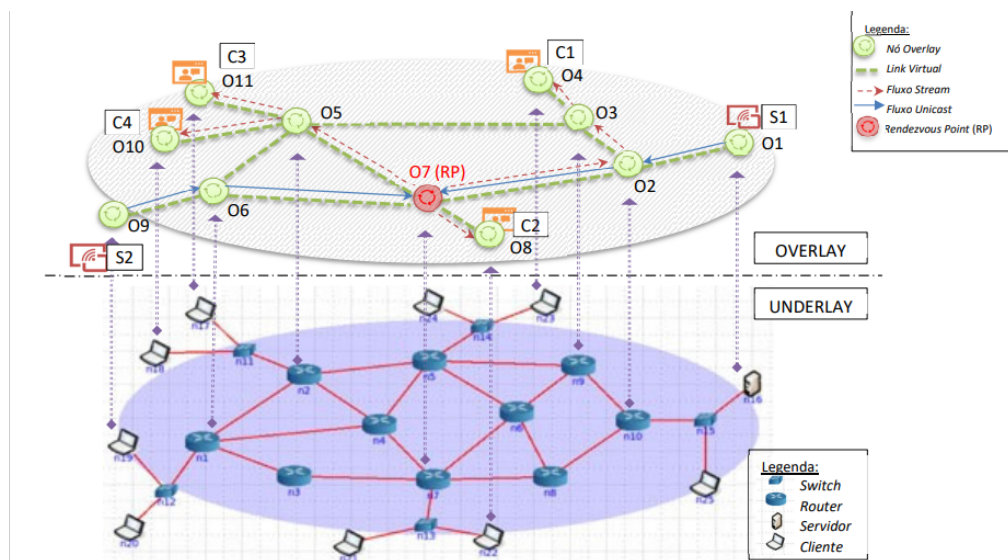


Figura 1: Visão geral de um serviço OTT sobre uma infraestrutura

## 2 Arquitetura da Solução

A arquitetura proposta para a solução consiste em clientes, e um servidor que atua como Rendezvous Point (RP), e nós overlay intermediários. Os clientes enviam pedidos ao RP, que avalia as condições dos servidores e seleciona o mais adequado. A transmissão do conteúdo é otimizada para reduzir a largura de banda usando o conceito de multicast. A rede overlay é construída sobre a rede underlay, e a árvore de distribuição é formada a partir dos nós folha. Durante a transmissão, o RP monitora continuamente o estado dos servidores, podendo realizar alterações dinâmicas conforme necessário.

### 2.1 Classes Constituintes

#### 2.1.1 RtpPacket

A classe **RtpPacket** desempenha um papel crucial na codificação e decodificação de pacotes RTP (Real-time Transport Protocol). Suas responsabilidades incluem a implementação de métodos que acessam e configuram os diversos campos do cabeçalho, assim como recuperam informações essenciais, tais como o número de sequência, o timestamp e o payload do pacote. O protocolo RTP, projetado para facilitar o transporte *end-to-end* de dados multimídia em tempo real por meio de redes, é suportado por esta classe. A utilização do mesmo abrange a criação de pacotes RTP, garantindo a configuração adequada dos campos do cabeçalho, além de analisar esses pacotes ao recebê-los.

#### 2.1.2 VideoStream

A classe **VideoStream** é responsável por ler o vídeo de um arquivo específico e recuperar a próxima frame após processar os primeiros 5 bits do arquivo, os quais indicam o comprimento da frame atual. Adicionalmente, esta classe incrementa o valor do comprimento lido. Além disso também gerencia o rastreamento do número da frame atual, inicializando-o com zero para indicar que ainda nenhuma frame foi lida.

#### 2.1.3 Client

A class **Client** desempenha um papel muito importante na interação com a entrega de conteúdo em tempo real através da rede overlay. Pois a mesma é responsável por iniciar o processo de comunicação com RP(Rendezvous Point) enviando pedidos. Esses pedidos indicam as preferências do cliente

em relação ao tipo específico de conteúdo que deseja receber, como vídeo áudio e textos...

#### **2.1.4 Server**

Essa classe é responsável por armazenar conteúdos e distribuí-lo para os clientes, pois ele atua como um provedor de conteúdo respondendo aos pedidos e garantindo a entrega em tempo real e com qualidade.

Inicialmente o server verifica quais são as streams que contêm e informa por unicast ao RP, de seguida fica a espera de pedidos do RP, que pode ser Setup, Play, Pause e Teardown.

#### **2.1.5 Nós**

A classe Nós é responsável por representar um nó em um sistema de streaming distribuído. A mesma possui métodos para analisar configurações a partir de um arquivo JSON, iniciar diferentes comportamentos com base no tipo do nó (Rp, Server, Client), e gerenciar conexões TCP. A classe implementa a lógica de um algoritmo de flooding para propagar mensagens em uma rede. Ela manipula o início e a espera de transmissões de vídeos, como o envio eficiente de pacotes de streaming entre os nós.

### **2.2 Rp**

Já a class RP desempenha um papel central, representando um nó do tipo Rp em um sistema de streaming distribuído. Suas principais responsabilidades incluem estabelecer e gerenciar conexões TCP, tratar mensagens recebidas, iniciar e controlar streaming de vídeo entre nós. Também implementa um protocolo de comunicação que inclui estados como: SETUP, PLAY E SETUP-REPLY, sendo capaz de enviar e receber streams de vídeos. A classe utiliza threads para operações concorrentes, incluindo a manipulação de sockets e envio eficiente de pacotes de streaming.

## **3 Especificação dos Protocolos**

Com base no objetivo do projeto, começamos por escolher a linguagem de programação Python, sendo a mais conveniente para o grupo, e posteriormente se definiu os protocolos de transportes TCP (Transmission Control Protocol) e UDP (User Datagram Protocol) a serem usados no overlay.

TCP : Definiu-se o protocolo TCP pela confiabilidade da transmissão dos conteúdos ou pacotes...

UDP: Definiu-se o UDP, por ser um protocolo ideal para transmissões de conteúdo em tempo real e pela sua rapidez ou tempo de resposta rápido e têm baixa latência, sendo isso que precisamos para streaming...

### 3.1 Formatos das Mensagens Protocolares

O conceito do protocolo de flooding é facilitar a determinação do melhor caminho para atender aos pedidos de streaming. Este protocolo é caracterizado por ser peer-to-peer, descentralizado e dinâmico. A implementação do mesmo tem por base o envio de mensagens. De maneira a auxiliar o processo de *flooding* criou-se um conjunto de variáveis locais na aplicação dos **Nós**, as quais desempenham um papel crucial neste processo.

No âmbito deste protocolo, é definido um dicionário denominado "request", que representa a mensagem enviada durante o processo de *flooding*. Este dicionário contém então as seguintes informações:

- **hostname**: IP do cliente.
- **stream\_name**: Nome do ficheiro do vídeo.
- **stream\_port**: IP da porta pelo qual a *stream* será feita.
- **state**: Tipo do pedido.
- **latencia**: Latência do cliente até ao nodo atual (calculada pela função *medir\_latencia*).
- **path**: Caminho do cliente até ao nodo atual ou RP (Trata-se dum array com todos os IPs da rota).
- **sequenceNumber**: Número de sequência do pacote RTP.

### 3.2 Interações

Com o objetivo de implementar uma rede *peer-to-peer* de maneira dinâmica, cada nó executará, durante a sua inicialização, um arquivo de inicialização do bootstraper, o qual irá realizar a configuração das respetivas variáveis locais. O cliente possui duas threads uma para executar os pedidos e outra para receber a stream. O nodo e o RP possuem também duas threads, uma para cada mensagem que recebe e outra para cada stream que passa pelo mesmo. Já o servidor possui uma thread para cada vídeo que tiver.

**Flooding** O cliente é quem começa o processo de Flooding enviando o pedido ao nodo mais proximo, de seguida os nodos reencaminham a mensagem

aos seus nodos vizinhos até chegar ao rp, quando chegar ao rp ele decidirá qual será o caminho pela nossa métrica(latência) e enviará a mensagem de novo para o cliente.

**Streaming** No processo de streaming, os nodos já sabem por onde têm que reencaminhar o streaming por causa do flooding que já foi feito anteriormente, então cada nodo só terá que abrir um socket udp para receber a stream anterior e abrir outro socket para enviar a stream para os próximos nodos.

## 4 Implementação

### 4.1 Construção da Topologia Overlay

O grupo de trabalho acabou por implementar a abordagem baseada num controlador *bootstrapper*, sugerida pela estratégia 3 da segunda etapa, para construir uma topologia overlay. Nesta estratégia sempre que se executa um cliente fornece-se como argumento a *tag* desse cliente no ficheiro do *bootstrapper*. Assim, sempre que um cliente é iniciado, este faz um pedido ao *bootstrapper*, uma vez que o mesmo inclui informações de todas as entidades constituintes da rede.

Em seguida apresentamos o exemplo das informações que o bootstrapper possui do cliente1, cuja *tag* é *cl1*:

```
"cl1":{
  "ip":"10.0.1.20",
  "type":"Client",
  "NO":["10.0.1.1"],
  "CL": [],
  "RP": "",
  "SV": []
}
```

### 4.2 Serviço de Streaming

No âmbito do serviço de streaming, o grupo desenvolveu um cliente e um servidor com base no código fornecido no livro de apoio pela equipe docente. O servidor é capaz de ler um vídeo de um arquivo e enviar seus dados em pacotes numerados para o cliente, que, por sua vez, reproduz o vídeo em uma nova janela. Nessa interface, quatro botões (SETUP, PLAY, PAUSE e TEARDOWN) são responsáveis por acionar solicitações específicas.



Para que uma solicitação seja transmitida ao servidor, é necessário associar cada tipo de solicitação a um estado anterior específico: *SETUP* → *INIT*; *PLAY* → *READY*; *PAUSE* → *PLAYING*; *TEARDOWN* → *NOTINIT*.

Ao receber o pacote RTP, o cliente transmite a imagem do vídeo em uma ordem predefinida, determinada pelo número de sequência de cada quadro, e não na ordem de chegada. Se o número de sequência atual for superior ao número do quadro, o pacote é descartado.

### 4.3 Monitorização dos Servidores de conteúdos

Na arquitetura do sistema, os servidores desempenham um papel crucial ao comunicar ao RP a lista de vídeos que possuem disponíveis.

Quando um cliente realiza flooding de um vídeo específico (setup), o RP entra em ação enviando uma mensagem de ping para os servidores que indicaram previamente possuir esse vídeo. O propósito desse ping é medir a latência entre o RP e cada servidor correspondente. Ao comparar essas latências, o RP escolhe o servidor que apresenta a menor latência para realizar a transmissão da stream ao cliente.

O grupo pensa que esta abordagem é altamente eficiente, pois permite ao RP tomar decisões informadas sobre qual servidor selecionar para otimizar a qualidade e a velocidade da transmissão. Ao priorizar o servidor com menor latência, o sistema proporciona uma experiência mais fluida e rápida para o cliente durante a reprodução do vídeo.

### 4.4 Construção dos Fluxos para Entrega de Dados

Através do processo de flooding previamente descrito, conseguimos estabelecer rotas para a entrega de dados. No entanto, é essencial determinar a rota específica para a entrega da stream solicitada.

Cada nó na overlay avalia a métrica de menor atraso, considerando também o critério de chegada mais rápida em casos de atrasos iguais. Com base nessa métrica, cada nó identifica o nó através do qual a entrega de dados será realizada. Esta abordagem visa medir a latência de nó a nó, somando-a à latência do pacote até atingir o RP. Quando um nó já possui a stream, a latência não é somada, otimizando o aproveitamento de nós que já recebem a stream solicitada.

## 5 Limitações da Solução

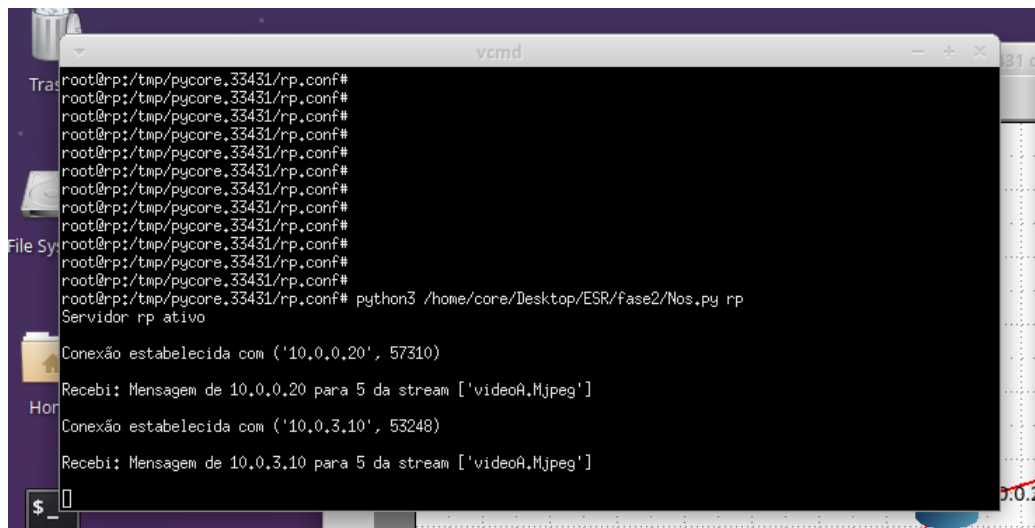
Não ter implementado uma forma de lidar com falhas e inclusão de novos nós em nossa solução é uma limitação da solução apresentada pelo grupo. Originalmente, assumimos que todos os nós estariam sempre disponíveis e que a composição da rede não mudaria. No entanto, na prática, nós podem falhar por vários motivos, como problemas de hardware ou conexões instáveis. A falta de um método para lidar automaticamente com essas falhas resulta, no nosso caso, em interrupções no serviço.

Além disso, não consideramos a possibilidade de novos nós se juntarem à rede, o que é comum em ambientes dinâmicos. A incapacidade de ajustar a estrutura para acomodar novos nós limita a flexibilidade do sistema.

Implementar estratégias simples para lidar com falhas e inclusão de novos nós seria benéfico para tornar a nossa solução mais robusta e adaptável a mudanças no ambiente de execução, garantindo um desempenho mais estável em situações do mundo real.

## 6 Testes e Resultados

Sempre que se iniciam os servidores, estes enviam mensagem ao RP a informar os vídeos que se encontram disponíveis. Isto comprova-se pelas seguintes prints do terminal do RP:



```
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf#
root@rp:/tmp/pycore.33431/rp.conf# python3 /home/core/Desktop/ESR/fase2/Nos.py rp
Servidor rp ativo
Conexão estabelecida com ('10.0.0.20', 57310)
Recebi: Mensagem de 10.0.0.20 para 5 da stream ['videoA.Mjpeg']
Conexão estabelecida com ('10.0.3.10', 53248)
Recebi: Mensagem de 10.0.3.10 para 5 da stream ['videoA.Mjpeg']
```

Figura 2: Inicialização dos servidores.

A Figura 3, ilustra um cliente a realizar flooding de um vídeo específico, neste caso do videoA.Mjpeg.

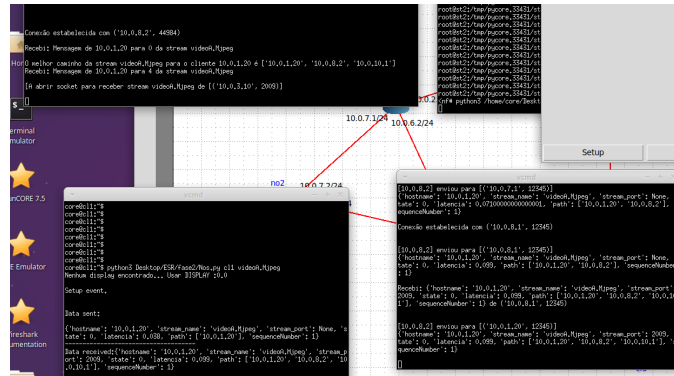


Figura 3: Flooding

Na Figura 4, encontra-se um stream de um mesmo vídeo a ser transmitido para 2 clientes diferentes, em simultâneo.

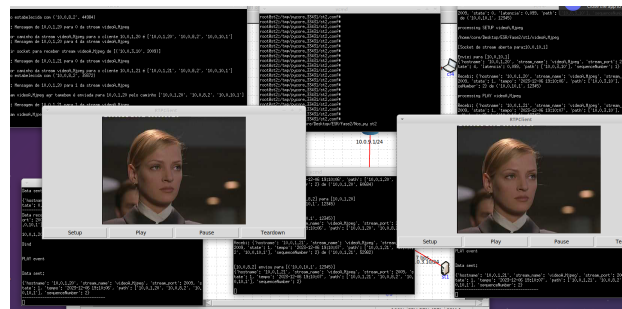


Figura 4: Stream de um vídeo para 2 clientes

Já na Figura 5, temos uma demonstração em que um cliente pausa um vídeo e o outro cliente não.

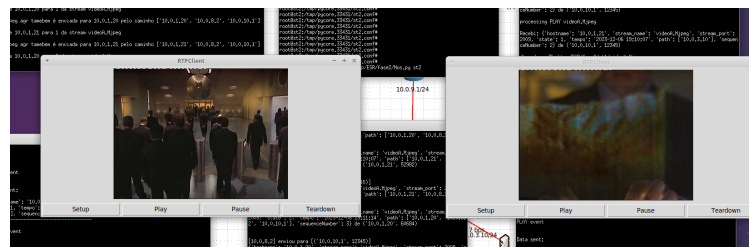


Figura 5: Stream para 2 clientes

## 7 Conclusão e Trabalhos Futuros

A conclusão deste projeto proporcionou uma consolidação significativa dos conceitos abordados nas aulas teóricas, especialmente no que diz respeito aos protocolos e serviços de streaming de vídeo.

O grupo de trabalho teve algumas dificuldades na resolução do projeto. Uma das maiores que surgiram foi em entender o que fazer na situação que dois clientes se encontram a ver o mesmo vídeo e em determinado momento um deles decide pausar o mesmo, sendo que a transmissão tem de continuar para o outro. Apesar de termos sido capazes de concluir este objetivo, de facto foi preciso dedicar algum tempo a entender como resolver este problema.

Por fim, gostaríamos ainda de ter sido capazes de implementar a etapa extra do guião do projeto, contudo o tempo para a realização do tal mostrou-se curto, por isso optamos por não implementar estas funcionalidades, com receio de também de deixar o projeto incompleto.

Esperemos que este projeto se mostre bastante relevante no nosso percurso profissional, uma vez que apreciámos desde o primeiro momento a resolução do mesmo.

# Bibliografia

URL <https://marco.uminho.pt/disciplinas/ESR/ProgEx.zip>