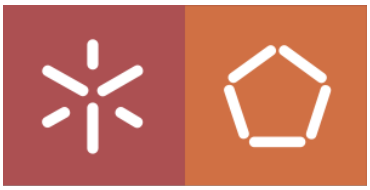

Universidade do Minho
Escola de Engenharia

Engenharia de Serviços em Rede

Serviço Over-the-top para entrega de multimédia

Grupo **PL13**

MEI - 1º Ano - 1º Semestre

Trabalho realizado por:

PG57867 - António Silva
PG55936 - Diogo Abreu
PG57872 - Duarte Leitão

Braga, 4 de dezembro de 2024

Índice

1. Introdução

Ao longo do último meio século de vida da Internet (a rede das redes), observou-se uma mudança irreversível de paradigma. A comunicação extremo-a-extremo, de sistema final para sistema final, dá lugar ao consumo voraz de conteúdos de qualquer tipo, a todo o instante, em contínuo e muitas vezes em tempo real. Este novo padrão de uso coloca grandes desafios à infraestrutura IP de base que a suporta.

Apesar de não ter sido originalmente desenhada com esse requisito, tem sido possível resolver a entrega massiva de conteúdos com redes sofisticadas de entrega de conteúdos (CDNs) e com serviços específicos, desenhados sobre a camada aplicacional, e por isso ditos Over-the-top (OTT). Um serviço de multimédia OTT, pode por exemplo usar uma rede overlay aplicacional (ex: multicast aplicacional), devidamente configurada e gerida para contornar os problemas de congestão e limitação de recursos da rede de suporte, entregando em tempo real e sem perda de qualidade os conteúdos diretamente ao cliente final.

Serviços bem conhecidos como o Netflix ou o Hulu, fazem streaming sobre a rede IP pública. Daí a designação de "Over-the-top streaming services". Para tal, formam uma rede overlay própria, assente em cima dos protocolos de transporte (TCP ou UDP) e/ou aplicacionais (HTTP) da Internet.

Neste trabalho, pretende-se conceber e prototipar um desses serviços, que promova a eficiência e a otimização de recursos para melhor qualidade de experiência do utilizador.

2. Arquitetura da solução

A solução proposta para este serviço está dividida em três componentes principais: Client, oNode e Server.

Em primeiro lugar temos o **Server**, que dispões de todo o conteúdo que é possível ser partilhado por stream. O **Client** neste serviço atua como um consumidor do conteúdo disponível na rede. Este começa por fazer um pedido de streaming e, caso e video pretendido esteja disponível, irá receber os pacotes e mostrar o video em tempo real. Para fazer a ligação entre clientes e servidor, existe uma rede de **oNode**. Estes componentes podem atuar tanto como cliente ou servidor, ou seja, tanto podem estar a receber a stream como também podem estar a difundir a mesma para outros oNodes ou clientes. No fundo, para uma stream chegar aos clientes, tem que passar pela rede de oNodes. Existe no entanto um mecanismo implementado nos oNodes que permite a cada um escolher de onde recebe a stream e assim fazer com que a stream percorra um caminho ideal até ao cliente.

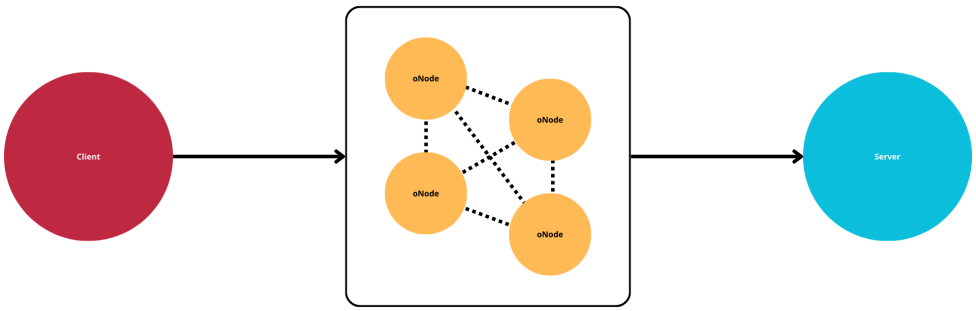

Figura 1: Arquitetura da solução

Esta arquitetura irá ser aplicada numa topologia abaixo. Na rede de acesso estão o clientes que vão consumir o conteúdo. No CDN temos vários oNodes que irão fazer a distribuição do conteúdo disponível no servidor. A fazer ligação entre a rede de acesso e o CDN, existem 3 oNodes de presença.

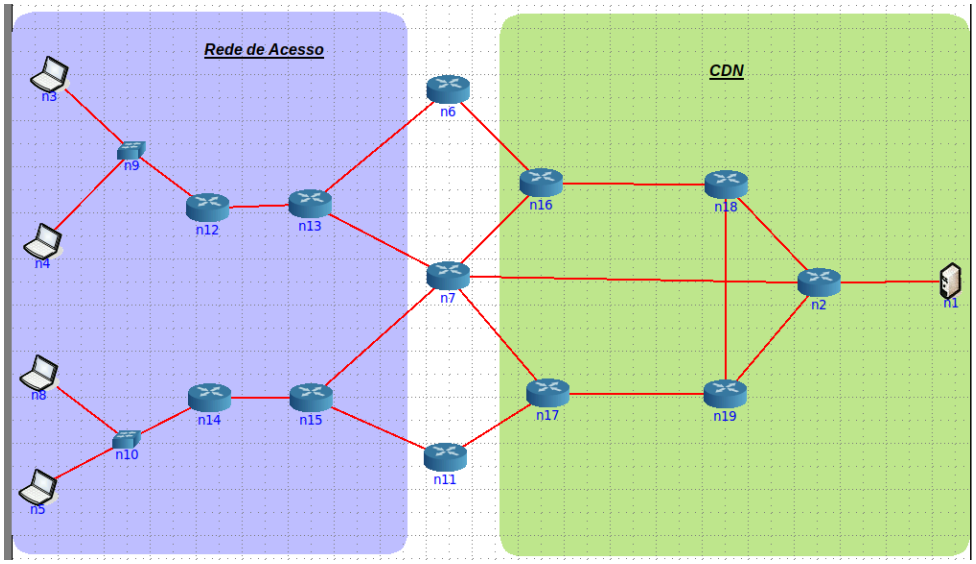

Figura 2: Topologia da rede

3. Especificação dos protocolos

3.1. Estrutura das mensagens

As mensagens trocadas entre clientes, nodos e servidor são criadas com recurso ao módulo *messages*. Este módulo contém funções que criam no formato desejado os vários tipos de mensagens bastando fornecer apenas os argumentos. Todas as mensagens contém um *id* seguido pelo tipo de mensagem. Existem 4 tipos de mensagens:

1. **CHECK_VIDEO**: usada para verificar se o vídeo está disponível para stream em algum dos elementos da topologia.
2. **READY**: usada durante o streaming para avisar um vizinho que está pronto a receber frames.
3. **DISCONNECT**: usada durante o streaming para avisar um vizinho que deve parar de enviar frames.
4. **PROBE**: as mensagens de probe são enviadas entre os vizinhos para obter métricas como o TTL e o delay entre eles. Isto é útil quando é necessário escolher o vizinho que têm a stream disponível.

3.2. Interações entre os componentes do sistema

3.2.1. Client e Server

Quando um cliente pretende obter a stream de um vídeo, estabelece uma conexão TCP com o servidor. O cliente envia um pedido inicial para o servidor, informando qual vídeo deseja. A resposta do servidor indica se o vídeo solicitado está disponível ou não. Se o vídeo for encontrado, o servidor inicia a stream do vídeo

3.2.2. Entre oNodes

Quando um cliente pede a stream um vídeo, o nodo primeiro verifica se ele próprio possui o vídeo ou se precisa de o pedir aos seus vizinhos. Isso é feito por meio da comunicação direta com os nodes vizinhos, utilizando sockets TCP.

Para garantir que o serviço funciona de forma eficiente, são trocadas mensagens de probe entre o nodos. Desta forma é possível tentar encontrar o melhor caminho para fazer chegar o vídeo ao cliente.

Quando um cliente pede um vídeo e o nodo local não tem esse vídeo armazenado, ele consulta os seus vizinhos. O processo de consulta aos vizinhos é realizado por meio de um protocolo de encaminhamento que pode envolver múltiplos nodos. Se um vizinho tiver o vídeo, ele pode então iniciar o processo de envio do vídeo para o cliente. Se o vídeo não for encontrado, o nodo pode passar o pedido para outros nodes ou até mesmo para o servidor.

A transferência de dados entre os nodos ocorre principalmente por meio de sockets UDP, quando um nodo envia pacotes RTP. Os nodos vizinhos podem atuar como difusores, recebendo pacotes e passando-os para outros nodos ou para clientes.

4. Implementação

4.1. Client

A classe principal, OClient, é responsável por gerenciar tanto a interface do utilizador quanto a comunicação com o servidor de vídeo.

Antes de iniciar a reprodução, o cliente verifica a disponibilidade do vídeo solicitado. A função *checkVideo* envia uma solicitação ao servidor para verificar se o vídeo está disponível. A função *requestVideo* envia uma mensagem ao servidor indicando que o cliente está pronto para receber o vídeo, iniciando o processo de recepção de frames.

A comunicação entre o cliente e o servidor é realizada através de sockets TCP e UDP:

- **Socket TCP**: Utilizado para enviar comandos e receber respostas do servidor. A conexão TCP é estabelecida com o servidor configurado como vizinho mais próximo, obtido através da função *Bootstrapper.get_neighbours*.
- **Socket UDP**: Utilizado para a transmissão dos frames de vídeo. O cliente configura o socket UDP para receber pacotes do servidor, que contêm os frames de vídeo codificados em pacotes RTP. A função *recieveFrame* recebe pacotes RTP, decodifica-os e extrai os frames de vídeo. Esses frames são então exibidos na interface gráfica.

A função *update_video_frame* atualiza a interface gráfica com os frames de vídeo recebidos, utilizando a biblioteca PIL para conversão e redimensionamento das imagens.

O OClient possui ainda a lógica para controlar a reprodução de video, incluindo controlos como Play, Pause e Stop.

4.2. Server

A implementação do servidor de streaming é centralizada na classe Server, que gerencia a comunicação com os clientes e distribui tarefas de processamento de vídeo.

Ao iniciar, o servidor cria um socket TCP, define opções de reutilização de endereço, e vincula o socket a uma porta específica. O servidor então entra em um loop de execução onde aceita conexões de clientes e cria uma nova thread para cada conexão, garantindo que o processamento de vídeo seja tratado de forma assíncrona e eficiente. Cada conexão é gerida por uma instância de *ServerWorker*, que se comunica com o cliente e realiza as operações necessárias.

A implementação da base de dados do servidor de streaming é realizada pela classe *ServerDatabase*. Esta classe é responsável por manter informações essenciais sobre os vídeos disponíveis, vizinhos do servidor, streams ativas dentro do servidor e mensagens processadas.

4.2.1. ServerWorker

A classe *ServerWorker* é responsável por gerir a comunicação com cada cliente conectado ao servidor de streaming, lidar com as solicitações de streaming de vídeo e enviar pacotes de vídeo RTP via UDP.

A função *startStream* é central para o funcionamento do *ServerWorker*. Este inicia o streaming de e entra num loop onde obtém frames de vídeo. Cada frame é enviado ao cliente via socket UDP num pacote RTP. O método também lida com possíveis erros na obtenção de frames e garante que a transmissão seja interrompida adequadamente ao fim do vídeo ou em caso de erro.

A função *run* gere a comunicação contínua com os clientes, processando os pedidos recebidos. A cada pedido, a mensagem é decodificada e as ações necessárias são executadas, como verificar a disponibilidade de um vídeo, preparar o streaming ou desconectar o cliente. Em caso de desconexão

inesperada, o método garante que o estado do streaming e a conexão sejam atualizados adequadamente, encerrando a transmissão de forma segura.

4.2.2. ServerDatabase

A classe *ServerDatabase* gere os dados essenciais para o funcionamento do servidor de streaming, incluindo a lista de vídeos disponíveis, vizinhos, o gerenciamento de streams ativas e as mensagens processadas.

Ao ser iniciada, a classe recebe um identificador único para o servidor e carrega a lista de vídeos disponíveis a partir da diretoria "videos". A partir daí, a cada vídeo é associado um objeto *VideoStream*, responsável pelo processamento e fornecimento dos frames para a transmissão. Além disso, a classe mantém um registro das streams ativas em um dicionário *videosStreaming*, que mapeia objetos *VideoStream* a instâncias de Streaming, representando o processo de transmissão de cada vídeo.

A classe também estabelece comunicação com os vizinhos do servidor, enviando "probes" periódicos para verificar o estado das conexões. O método *sendProbes* envia uma solicitação de "probe" para cada vizinho.

Além disso, a *ServerDatabase* possui ainda funções para gerir o fluxo de dados para os clientes. O método *enableStream* permite iniciar uma nova transmissão de vídeo, criando uma instância de Streaming para o vídeo solicitado, e assegura que o servidor mantenha o controlo sobre as streams em andamento. Caso o vídeo já esteja a ser transmitido, o método apenas conecta o novo cliente à transmissão existente.

4.3. oNode

A classe *ONode* é responsável por gerir a operação de um nó que atua como cliente e servidor, permitindo a comunicação com outros nós e o processamento de requisições. Ao ser inicializada, a classe cria um socket TCP e começa a escutar por conexões de entrada. Uma vez que uma conexão é estabelecida, o nó aceita a conexão e cria uma nova instância da classe *NodeWorker*, que é responsável por gerir a comunicação com o cliente. Cada cliente é tratado de forma independente em uma thread separada. Esta classe também tém uma *NodeDatabase*, que é utilizada para gerir os dados e operações do nó. O base de dados do nó armazena informações relevantes, como vizinhos, delays, steps e streams ativas.

4.3.1. NodeWorker

A classe *NodeWorker* é responsável por gerir a comunicação entre um nó e os clientes conectados. Esta classe encarrega-se de lidar com pedidos de streaming, enviar pacotes RTP via UDP e manter a conexão ativa com os clientes.

A função *startStream* é responsável por iniciar o streaming do vídeo pedido pelo cliente. Primeiro, verifica se a transmissão do vídeo pode ser iniciada chamando a função *enableStream*. Se a stream for iniciada, começa o envio de frames do vídeo ao cliente. Cada frame é empacotado num pacote RTP e enviado através do socket UDP. A função também implementa uma lógica que, caso ocorra um erro ao obter um frame, tenta novamente até 10 vezes antes de cancelar a operação. Durante a transmissão, é incrementado o número de sequência para cada frame enviado.

A função *checkVideoOnSystem* verifica se o vídeo pedido pelo cliente está disponível no sistema. Caso o vídeo não esteja disponível localmente, o nó vai perguntar aos nós vizinhos se algum deles possui o vídeo.

4.3.2. NodeDatabase

A classe *NodeDatabase* desempenha um papel fundamental na gestão das operações de streaming de um nó. Esta classe é responsável por manter informações sobre os nós vizinhos, gerir vídeos em streaming, e lidar com a distribuição de mensagens de probe e visualização de mensagens.

No momento da inicialização, a classe *NodeDatabase* obtém a lista de nós vizinhos chamando a função *get_neighbours* do *Bootstrapper*. lém disso, a classe inicializa outras variáveis importantes, como *neighboursDelay*, que armazena os atrasos de comunicação com os vizinhos, e *viewedMessages*, que mantém um registro das mensagens já vistas para evitar processamento redundante.

A função *checkVideoOnSystem* verifica se o vídeo pedido pelo cliente está disponível no sistema. Se o vídeo não estiver a ser transmitido pelo nó atual, ele tenta pedir o vídeo aos nós vizinhos. Esta função envia o pedido original a cada vizinho e aguarda uma resposta positiva.

A função *enableStream* tenta iniciar a transmissão de um vídeo solicitado. Caso o video se encontre disponível no nó em questão, esta função apenas conecta o utilizador á stream existente. Caso a stream tenha que ser pedida a outro nó existe um conjunto de critérios que devem ser tidos em conta para escolher qual dos nós (que dispõe da stream) deve ser escolhido:

1. Primeiro, os vizinhos são ordenados pelo **atraso de comunicação** registrado em neighboursDelay. Este atraso é calculado com base no tempo de resposta das mensagens de probing recebidas.
2. Em seguida, os vizinhos são ordenados pelo número de saltos **TTL (Time-To-Live)**, que indica a proximidade em termos de rede. Vizinhos com menor TTL são preferidos.

A função *handleProbing* lida com as mensagens de probe recebidas de outros nós. Esta incrementa o TTL (Time-To-Live) e atualiza os atrasos de comunicação com os vizinhos.

5. Testes e Resultados

Durante e após a implementação de todo o código realizamos vários testes de forma a avaliarmos o desempenho da solução e a sua eficiência na apresentação de todo o conteúdo multimédia.

Os dois cenários principais que conseguimos diferenciar é a existência de um único conteúdo para diversos clientes e da existência de dois diferentes conteúdos para diferentes clientes.

No primeiro caso, um ponto crítico que conseguimos verificar é que os nossos Pontos de Presença (PoPs) guardam em cache o conteúdo multimédia que está a ser produzido. Isso acontece pois, como conseguimos ver, quando o segundo cliente solicita o mesmo vídeo que está a ser produzido para o primeiro cliente, o conteúdo é entregue diretamente pelo PoPs que já o continha em cache, evitando que o pedido feito pelo segundo cliente tenha a necessidade de comunicar com o servidor, reduzinho a carga que passa pela infraestrutura e a latência que existe para o segundo cliente. Além disso, conseguimos verificar que o conteúdo reproduzido encontra-se a ser apresentado em ambos os clientes no mesmo instante.

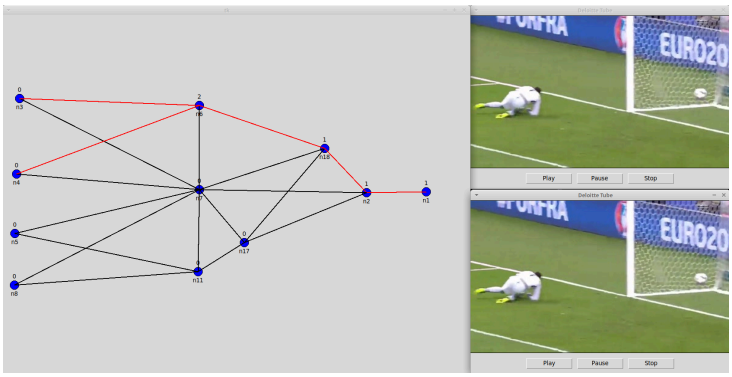

Figura 3: Stream de um conteúdo

No segundo cenário, o servidor distribui dois conteúdos diferentes ao mesmo tempo, demonstrando que a nossa solução consegue produzir mais do que um conteúdo ao mesmo tempo. Neste caso conseguimos ver que no servidor tem o número 2, representando os 2 tipos de conteúdo, tal como no node 2. Já no resto dos nodos onde só passa uma vez o conteúdo só tem o número 1.

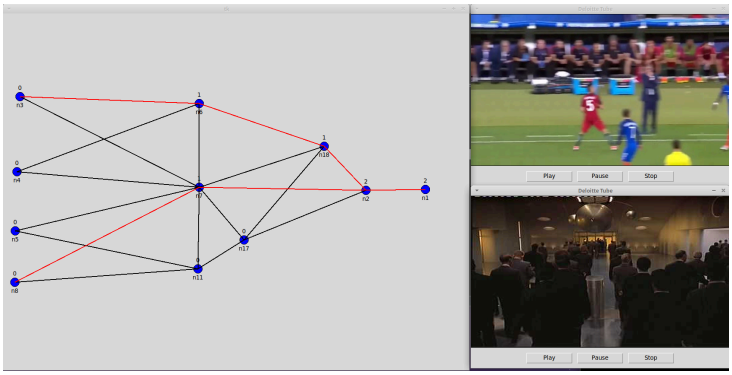

Figura 4: Stream de dois conteúdos em simultâneo

Além destes cenários com diferente quantidade de conteúdo, também testamos o que aconteceria se adicionássemos um delay numa conexão entre dois nodos. Colocamos um delay de 200 ms entre o n2 e o n7 para testarmos como o programa se iria comportar com algo desse gênero. Neste teste só tinhamos ligados os nodos n7, n18 e n2, além do server (n1) e o cliente (n13). Apesar do número de saltos ser maior, devido ao facto da conexão entre o n2 e o n7 estar lenta, a stream do conteúdo utiliza um caminho alternativo para continuar a ter uma exibição de conteúdo fluída.

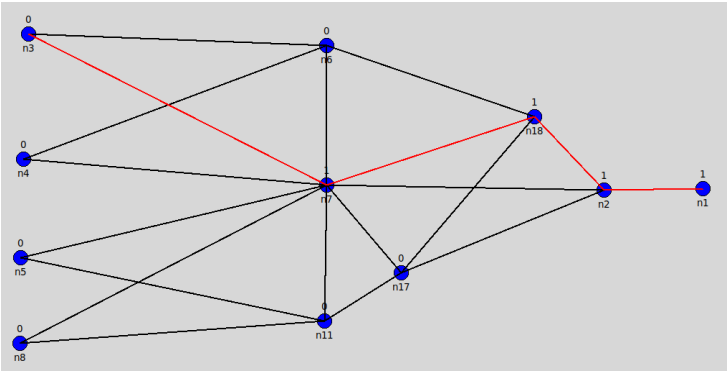

Figura 5: Stream de conteúdo com delay no caminho

Com estes testes, conseguimos comprovar que a nossa solução consegue reproduzir vários tipos de conteúdo de forma eficiente e o facto de os PoPs terem cache faz com que o programa seja mais eficiente e haja menos carga no servidor. Também conseguimos comprovar que temos as nossas métricas a funcionar de forma eficaz no programa escolhendo sempre o caminho com melhor qualidade, avaliando quer o número de saltos, quer a largura de banda.

6. Conclusão

Durante a produção deste projeto, fomos concebendo e implementando um Serviço Over-the-Top (OTT) com objetivo para a entrega de multimédia, sendo destacado pela utilização de uma arquitetura que foi otimizada para a distribuição de conteúdo em tempo real. A solução que produzimos combina os vários princípios de eficiência e escalabilidade, enfrentando todos os desafios inerentes à infraestrutura da Internet para o consumo massivo de conteúdos.

Esta implementação acabou por ficar dividida em três componentes principais que já explicamos: o Client, o Server e o oNode, sendo que estes estão todos interligados por uma Rede Overlay Aplicacional.

Os testes que realizamos confirmaram o cumprimento dos requisitos definidos no enunciado, destacando-se a medição de atrasos, perdas e a capacidade de resposta da rede a diferentes cenários dinâmicos.

No entanto, o trabalho ainda se encontra com capacidades de evolução. Embora todos os requisitos mínimos tenham sido cumpridos, diversas funcionalidades adicionais, como o suporte a múltiplos servidores e adaptação dinâmica às condições de rede durante o *streaming*, poderiam ser exploradas de forma a aumentar a robustez e a versatibilidade do programa.

Resumindo, o trabalho alcançou os objetivos propostos, evidenciando a viabilidade de uma arquitetura OTT eficiente para entrega de conteúdos multimédia.