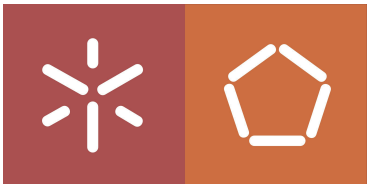

Universidade do Minho
Departamento de Informática

Engenharia de Serviços em Rede

Ano Letivo 2024/2025

Trabalho Prático N°2

Grupo 2

Lucas Oliveira - PG57886

Rafael Gomes - PG56000

Rodrigo Casal Novo - PG56006

Índice

1. Introdução

Este relatório descreve o desenvolvimento de um serviço **Over-the-Top** (OTT) para entrega de conteúdos multimédia em tempo real, realizado no âmbito da unidade curricular de Engenharia de Serviços em Rede. O objetivo principal deste trabalho foi conceber e prototipar uma solução que utiliza uma rede overlay aplicacional e pontos de interligação (Points of Presence) para otimizar a entrega de vídeo com requisitos rigorosos de qualidade e eficiência.

Ao longo do trabalho, foram desenvolvidas diversas etapas, incluindo a construção de uma topologia de rede overlay, a implementação do serviço de streaming, a monitorização de servidores de conteúdos, e a configuração de fluxos de dados. Adicionalmente, foi definida uma estratégia para gestão de falhas e integração de novos nós na rede, garantindo robustez e escalabilidade do sistema.

O relatório detalha todas as fases do desenvolvimento, abrangendo a conceção inicial, implementação técnica, realização de testes, análise dos resultados obtidos, conclusões extraídas e propostas para trabalho futuro.

2. Arquitetura da solução

A arquitetura da solução divide-se em **duas principais** camadas de rede: a **rede *CDN*** (*Content Delivery Network*) e a **rede de acesso**. Dentro da rede *CDN*, nós intermediários formam uma rede *overlay* para otimizar a entrega de conteúdos, reduzindo atrasos e minimizando o consumo de largura de banda. Na rede de acesso, os clientes consomem os conteúdos por via de conexões *unicast* estabelecidas a partir de **Pontos de Presença** (*PoPs*), que fazem a ponte entre a rede *CDN* e os utilizadores finais.

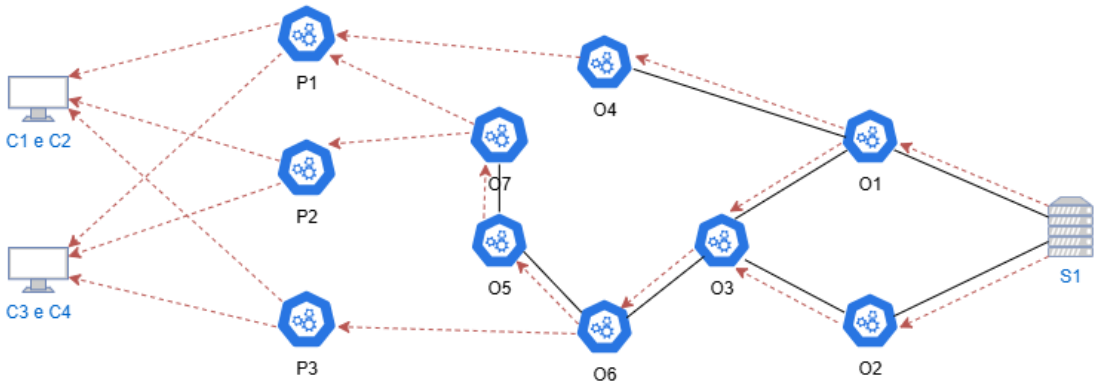

Figura 1: Rede Overlay.

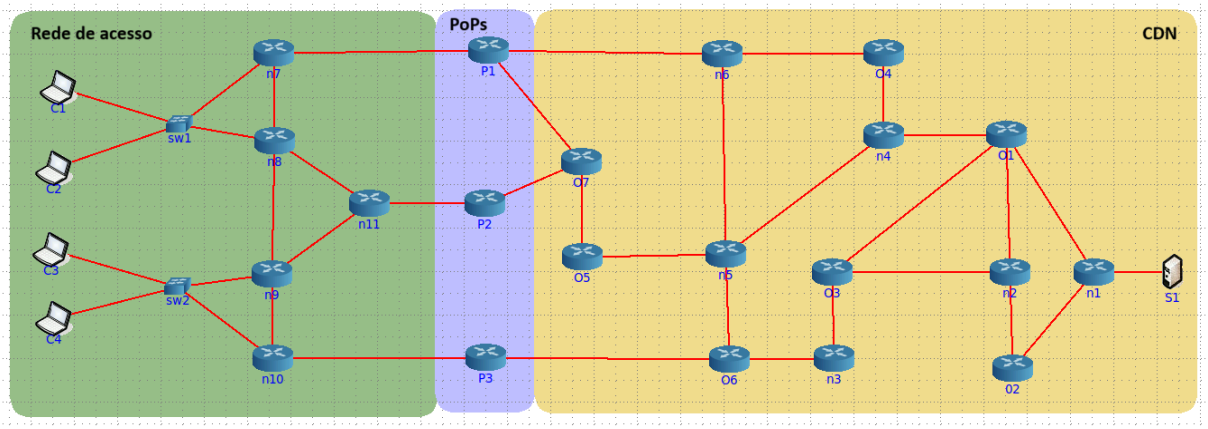

Figura 2: Rede Underlay.

Para a implementação do trabalho prático, optamos pela utilização da linguagem de programação ***Python***, uma vez que oferece um elevado nível de abstração e disponibiliza uma vasta gama de bibliotecas que simplificam a manipulação de **sockets** e **threads**. Além disso, trata-se da linguagem com a qual os membros da equipa têm maior familiaridade e conforto.
A estratégia selecionada para a construção da rede overlay foi a estratégia 2, utilizando o método de ***bootstrapper*** para fornecer os vizinhos de cada nó. Esta escolha deve-se ao seu maior dinamismo e escalabilidade, permitindo a criação automatizada da rede com base num nó de contacto inicial.

3. Especificação dos protocolos

Selecionamos o ***RTP*** (*Real-time Transport Protocol*) como o nosso protocolo aplicacional para a realização do *streaming*, tendo aproveitado uma implementação fornecida pela equipa docente. Este protocolo realiza o envio de mensagens por meio do protocolo de transporte *UDP*, conhecido pela simplicidade e capacidade de operar com baixa latência, características indispensáveis para transmissões em tempo real. O *RTP*, sendo um protocolo desenvolvido especificamente para aplicações de *streaming*, revelou-se uma escolha adequada para o nosso projeto, assegurando uma entrega eficiente de conteúdos multimédia mesmo em cenários onde há tolerância a perdas de pacotes.

Para a comunicação entre os diversos componentes da infraestrutura, adotamos abordagens distintas para otimizar o desempenho. A comunicação entre os *Points of Presence* (*PoPs*) e os clientes utilizou o protocolo *UDP*, priorizando baixa latência e eficiência. Entre o servidor de conteúdos e os nós da rede *overlay* (*Os*), também utilizamos *UDP*, mantendo a simplicidade e alinhando-se com o protocolo *RTP* escolhido para o *streaming*.

Já entre os nós do *overlay* (*Os*), optamos pelo protocolo de transporte *TCP*, devido à necessidade de conexões confiáveis para a construção e manutenção das tabelas de rotas e para a coordenação eficaz do tráfego dentro da rede *overlay*. Essa combinação de protocolos: *UDP* para o *streaming* e comunicação com os clientes, e *TCP* para a comunicação interna dos nós do *overlay* — garantiu uma solução robusta, eficiente e alinhada com os requisitos de baixa latência e confiabilidade do projeto.

4. Implementação

Primeiramente, antes da implementação das etapas, foi estruturada uma classe mensagem, com o intuito de facilitar as comunicações na nossa rede.

4.1. Mensagem

As mensagens utilizadas na comunicação entre os vários intermediários da rede possuem os seguintes campos:

- **type**:
  - **START_SERVICE (1)** : Utilizado para quando um nodo é inicializado, avisar o servidor e receber os seus vizinhos.
  - **SEND_NEIGHBORS (3)** : Utilizado para enviar os vizinhos dos respetivos nodos.
  - **REQUEST_STREAM (4)** : Utilizado para fazer pedido de *streaming* de um video.
  - **NOTIFY_NEIGHBOR (6)** : Utilizado para notificar um nodo caso tenha perdido conexão com o seu vizinho.
  - **FLOODING (7)** : Utilizado para fazer o *flooding* da rede.
  - **START_VIDEO (9)** : Utilizado para notificar todos os nodos do inicio de uma *stream*.
  - **METRICS (10)** : Utilizado para calcular o melhor *PoP* para cada cliente.
  - **END_STREAM (11)** : Utilizado para pedir de parar de receber uma *stream*.
  - **ACK_END_STREAM (12)** : Utilizado para confirmar o fim de uma *stream*.
- **data**: O conteúdo da mensagem.
- **sender**: Quem enviou a mensagem.

- **dest**: O destinatário da mensagem.

4.2. Etapa 1: Construção da Topologia Overlay e Underlay

Conforme referido anteriormente, optámos pela estratégia do *bootstrapper* para a construção da rede *overlay*.

O processo inicia com o servidor a carregar, no arranque, um ficheiro *JSON* que contém a configuração da rede, especificando os vizinhos de cada nó. Esta informação é armazenada pelo servidor e utilizada para responder a pedidos de novos nós que queiram integrar a rede.

Após o servidor estar ativo, quando um novo nó é iniciado, este conecta-se ao servidor via **protocolo TCP** e envia uma mensagem do tipo **START_SERVICE**. Em seguida, aguarda pela resposta do servidor com a lista dos seus vizinhos na rede. O servidor, ao receber a mensagem, identifica o nó através do seu endereço IP, mapeado por DNS, e utiliza essa identificação para localizar os vizinhos atribuídos a esse nó.

O servidor, então, constrói uma mensagem do tipo **SEND_NEIGHBORS**, que inclui uma lista de tuplos. Cada tuplo contém o IP de um vizinho e uma flag que indica se esse vizinho já está ativo na rede. Esta mensagem é enviada ao nó solicitante.

Quando o nó recebe a mensagem do servidor, analisa a lista de vizinhos. Para os vizinhos marcados como ativos na flag, o nó cria uma thread dedicada a estabelecer e gerir a conexão entre si e esses vizinhos, permitindo a formação gradual e estruturada da rede overlay.

4.3. Etapa 2: Construção oClient

No processo de conexão do cliente ao sistema, o cliente inicialmente estabelece uma comunicação com o servidor, que lhe fornece informações sobre os **Points of Presence (PoPs)** disponíveis. Todas as interações entre os clientes e os *PoPs* são realizadas através do protocolo de transporte *UDP*, visando manter uma comunicação eficiente e de baixa latência, adequada para o cenário de *streaming* em tempo real.

A seleção do **PoP** ideal para o cliente solicitar o pedido de *streaming* do conteúdo segue um conjunto de métricas bem definidas. Primeiramente, caso um *PoP* **já esteja a transmitir o vídeo solicitado** pelo cliente, esse *PoP* é automaticamente escolhido para a entrega do conteúdo, otimizando recursos e minimizando atrasos.

Se nenhum *PoP* estiver a transmitir o vídeo desejado, o cliente utiliza uma métrica **baseada no tempo de resposta** para determinar o *PoP* mais eficiente. Nesse processo, o cliente envia cinco mensagens de teste para cada *PoP* disponível e calcula a média dos tempos de resposta. O *PoP* com o menor tempo médio é então selecionado para fornecer o vídeo. Essa métrica é recalculada a cada 5 segundos, garantindo que o cliente tenha sempre acesso ao *PoP* mais rápido, mesmo em caso de alterações nas condições da rede.

Utilizando assim este mecanismo de seleção, conseguimos assegurar uma melhor experiência para o utlizador e uma melhor gestão dos recursos da rede.

Após o cliente conectar-se ao sistema, são apresentadas as opções disponíveis para interação, conforme ilustrado na Figura 3. Essas opções permitem ao utilizador navegar pelas funcionalidades do sistema de forma intuitiva e prática:

- **Pedir video ao servidor:** o cliente pode visualizar a lista de vídeos atualmente disponíveis no servidor. Após a apresentação da lista, o utilizador pode selecionar o vídeo que deseja solicitar para transmissão.
- **Conhecer vizinhos:** esta opção exibe os vizinhos do cliente na rede, indicando qual deles apresenta o melhor tempo de resposta. Além disso, são listados os vídeos que cada um desses

vizinhos já está a transmitir no momento, proporcionando ao cliente informações úteis para tomar decisões de consumo de conteúdo de forma eficiente.

- **Desconectar:** o cliente pode optar por desconectar-se da rede, encerrando sua conexão no sistema de forma correta.

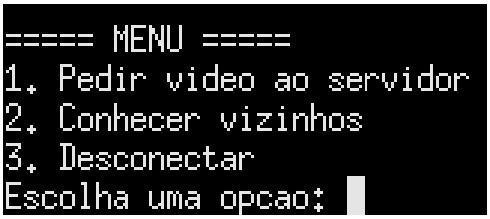

Figura 3: Menu Cliente.

4.4. Etapa 3: Serviço de Streaming

Nesta terceira etapa da nossa implementação foi necessário implementar novas funcionalidades em todos os programas da nossa topologia.

4.4.1. Serviço de Streaming nos Nodos intermédios

Numa fase inicial desta etapa apontamos o nosso foco à passagem da mensagem, desde o PoP escolhido pelo cliente até, no pior cenário, ao servidor, e a resposta do mesmo, (com protocolo REQUEST_STREAM e START_VIDEO respetivamente).

A mensagem com o pedido tem como objetivo preparar a sub-rede UDP para transmissão dos pacotes RTP e, caso haja já uma transmissão a decorrer do respetivo vídeo no nodo, o próprio nodo irá replicar a transmissão para o novo pedido.

Quando é pedido o término da transmissão por parte do nodo anterior, cabe a cada nó interromper a transmissão de pacotes e informar o nodo seguinte, quando não restar pedidos de streaming.

4.4.2. Serviço de Streaming no Cliente

Este programa, após receber o pedido por parte do utilizador, envia o mesmo ao PoP escolhido e prepara-se para receber pacotes. O utilizador,para além de continuar com as mesmas funcionalidades, pode ainda pausar e retomar a transmissão, e terminar a transmissão.

Quando o utilizador pede para terminar a transmissão, há semelhança do pedido para a iniciar, este é enviado ao PoP.

4.4.3. Serviço de Streaming no Servidor

O servidor quando recebe o primeiro pedido de transmissão de um vídeo, seleciona o pacote a ser enviado, constrói a mensagem RTP e envia a mesma pela sub-rede construida pelos nodos intermédios. Enquanto isso, envia pela rede overlay uma mensagem do tipo START_VIDEO.

4.5. Etapa 4: Construção das Árvores de Distribuição

Nesta etapa, identificou-se a necessidade de que cada nodo fosse capaz de orientar mensagens dentro da topologia sem a necessidade constante de realizar broadcasts. Para alcançar este objetivo, foi essencial que cada nodo criasse e populasse adequadamente a sua própria tabela de endereçamento.

Dado que a preparação da rede não estará completamente pronta antes da inicialização dos nodos de presença - os quais representam folhas na topologia, decidiu-se que a melhor abordagem seria iniciar o processo de *flooding* a partir destes nodos no momento de sua ativação. Essa ação seria executada apenas quando todos os nodos intermédios vizinhos já estivessem devidamente iniciados.

Esta mensagem tem como objetivo preencher ao máximo as tabelas dos nodos intermédios e por isso faz um broadcast controlado com o servidor como destino e, no conteúdo leva a lista de nodos já

visitados. No servidor é escolhido o melhor caminho com base no numero de saltos e, em caso de empate a ordem de chegada.

Visando manter uma topologia bem distribuída e eficiente, este processo de flooding é repetido em intervalos de 150 segundos, garantindo que as tabelas de endereçamento permaneçam atualizadas frente a alterações na rede.

4.6. Extra: Definição do método de recuperação de falhas

Para simular falhas no sistema, partimos do princípio de que qualquer comunicação que dependesse de caminhos envolvendo um nodo em falha seria automaticamente inviabilizada. Assim, seria imprescindível realizar uma reconstrução para manter o funcionamento normal da rede.

Nesse contexto, determinamos que seria necessária uma reconstrução dinâmica da árvore de endereçamento sempre que um nodo falhasse, garantindo que a comunicação e os serviços dependentes da topologia fossem restabelecidos de forma eficiente.

4.6.1. Implementação

Durante a inicialização dos pontos de presença, uma vez que estes são os responsáveis por iniciar os processos de flooding, cada nodo estabelece também uma conexão com um socket de notificação (notify). Essa conexão tem como único propósito informar os nodos de presença sobre a perda de conexão de qualquer nodo intermediário com o bootstrapper da rede.

Ao receber essa notificação, os nodos de presença iniciam um broadcast extraordinário para propagar as informações atualizadas na rede. Este broadcast serve para reconstruir a árvore de endereçamento, garantindo que novos caminhos sejam definidos para manter a conectividade entre os nodos de presença e o servidor, minimizando o impacto das falhas e restabelecendo a comunicação essencial de forma eficiente.

5. Testes e Resultados

Para os testes demonstrados nesta secção foram iniciados os elementos apresentados na seguinte figura:

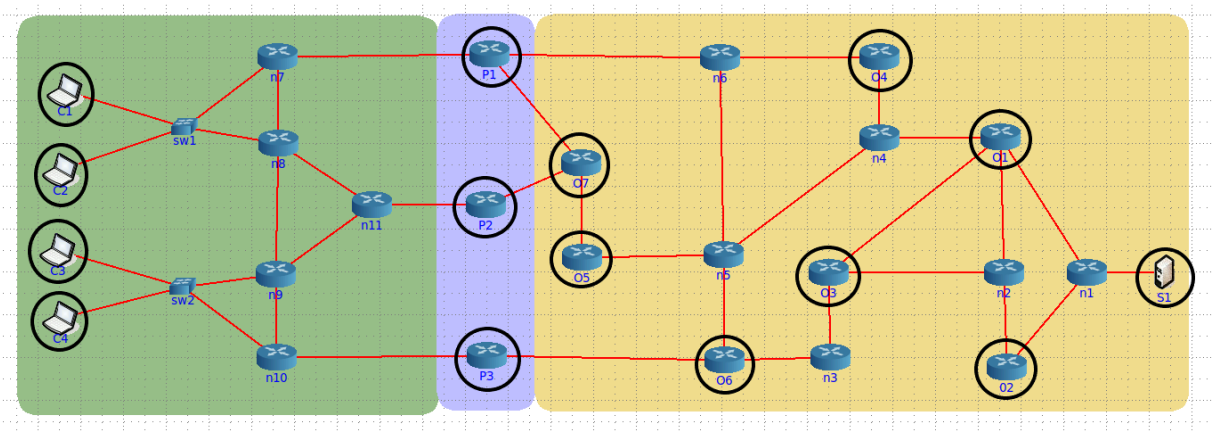

Figura 4: Elementos da Topologia utilizados.

5.1. Streaming de Video

5.1.1. Transmissão do mesmo vídeo para dois clientes

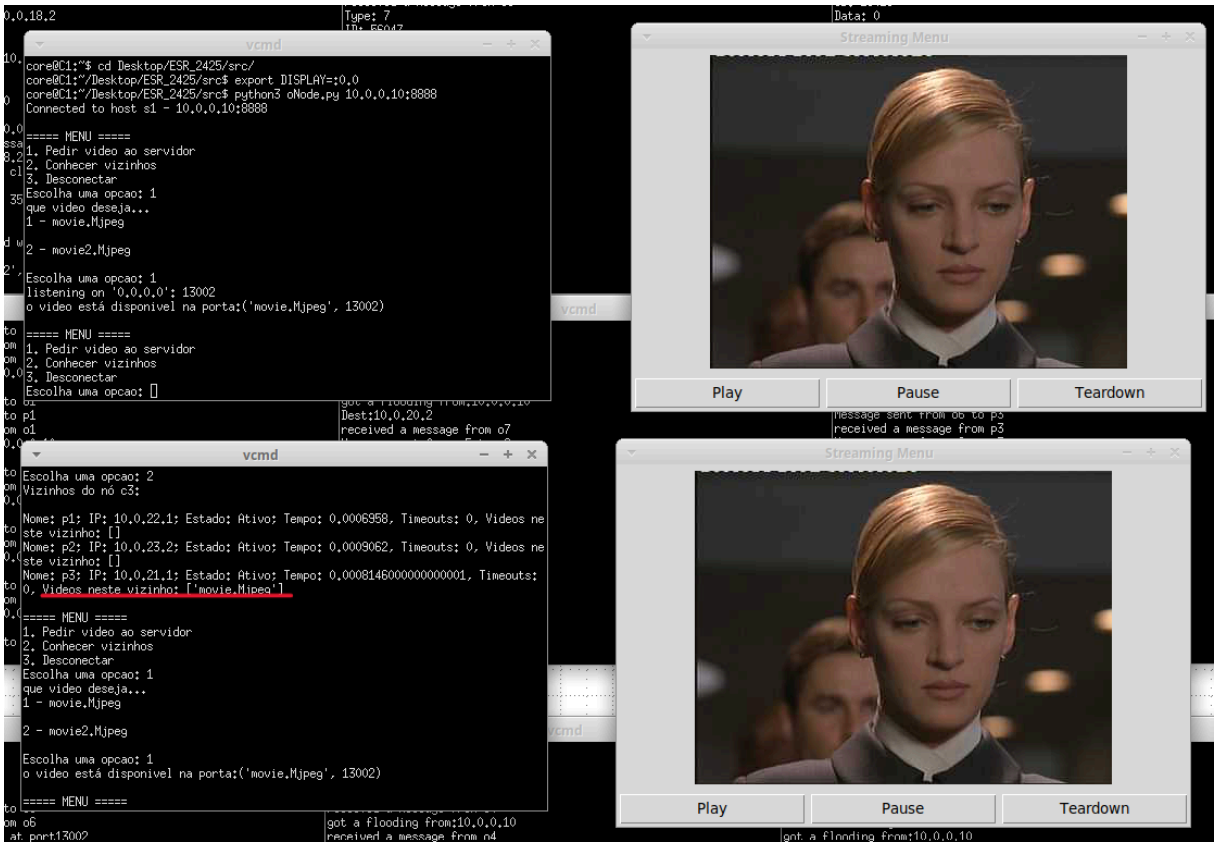

Figura 5: Transmissão para dois clientes.

Conforme ilustrado na Figura 5, o cliente *C1* foi o primeiro a estabelecer conexão e, em seguida, solicitou o vídeo *"movie.Mjpeg"*, que foi pedido ao servidor e após isso apresentado. Posteriormente, o cliente *C3* realizou a sua conexão e, podendo verificar no seu terminal que o vídeo "movie.Mjpeg" já estava em transmissão, realizou a solicitação do mesmo. O vídeo foi então fornecido imediatamente pelo *PoP* responsável pela sua transmissão.

5.1.2. Transmissão de dois vídeos diferentes para o mesmo cliente

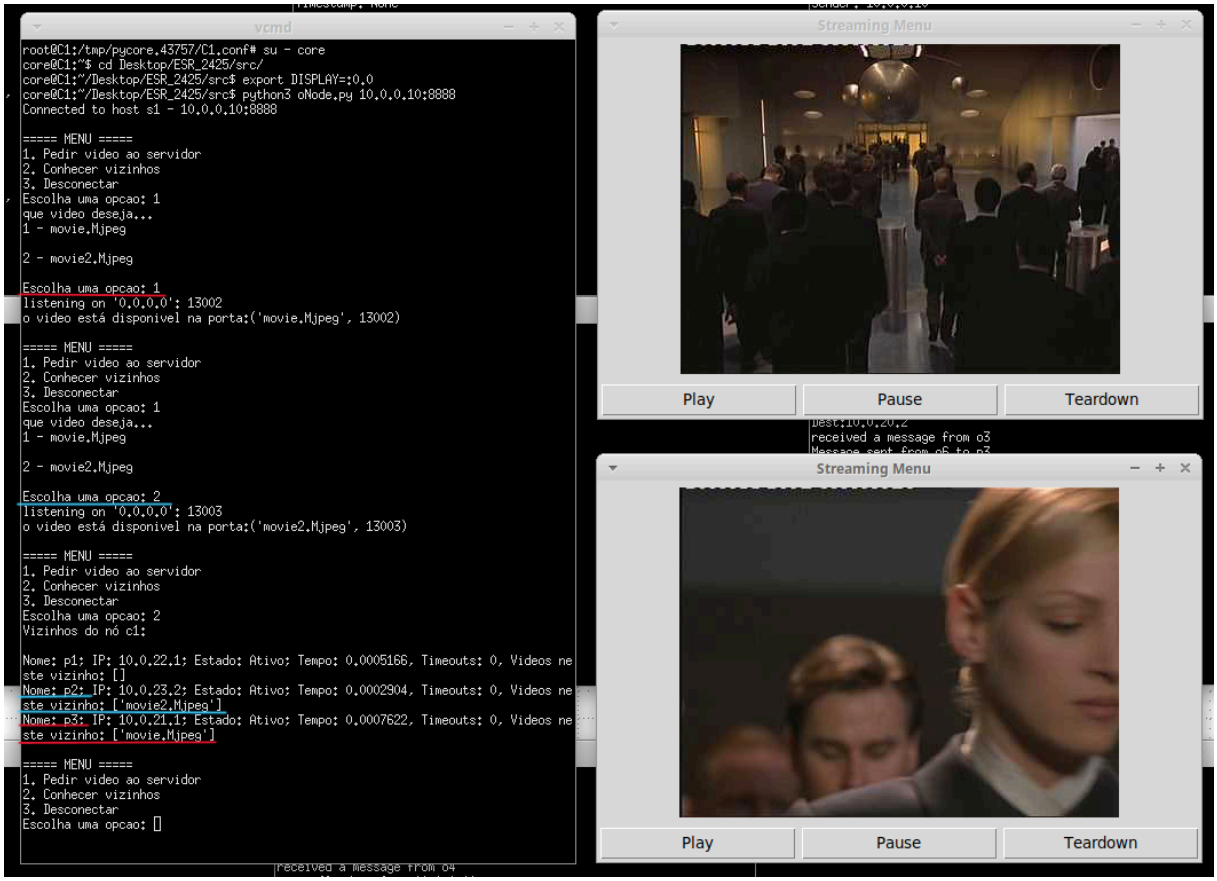

Figura 6: Transmissão de dois videos para um cliente.

Também é possível que um mesmo cliente solicite dois vídeos diferentes, como ilustrado na Figura 6. Podemos observar que o cliente *C1* estabeleceu a conexão e, em seguida, pediu o vídeo *"movie.Mjpeg"*. Logo após, fez a solicitação do vídeo *"movie2.Mjpeg"*. Esses dois conteúdos foram entregues por *PoPs* distintos, demonstrando a capacidade da rede em distribuir diferentes fluxos simultaneamente para o mesmo cliente.

5.1.3. Transmissão de um vídeo quando um dos nodos de transmissão falha

Como parte de uma etapa complementar, analisamos o comportamento da rede em cenários de falha.

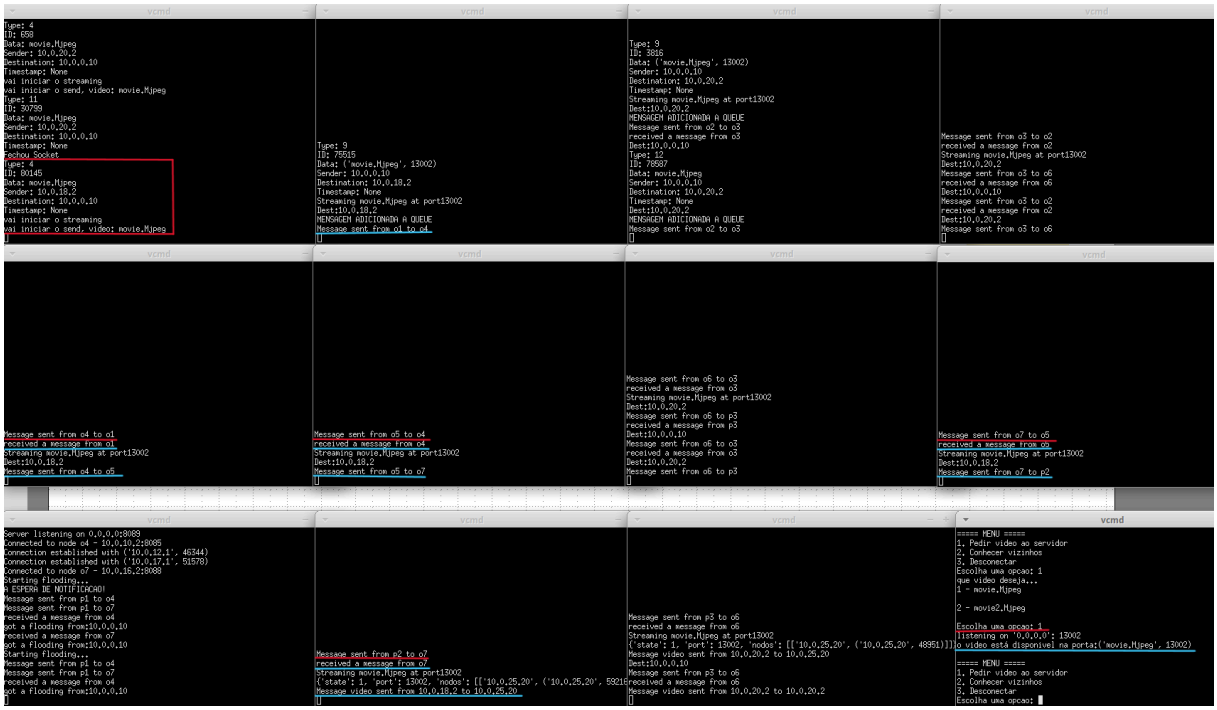

Figura 7: Transmissão de dois videos para um cliente.

Inicialmente, o percurso do pedido do conteúdo seguiu a rota: *P2 -> O7 -> O5 -> O4 -> O1 -> S1*, como podemos ver na Figura 7 sublinhado a *vermelho* o caminho do pedido da *stream* do vídeo e a *azul* a resposta do mesmo). No entanto, ao ocorrer a desconexão do nodo *O1*, que estava diretamente conectado ao servidor, o sistema reconstruiu a árvore de distribuição.

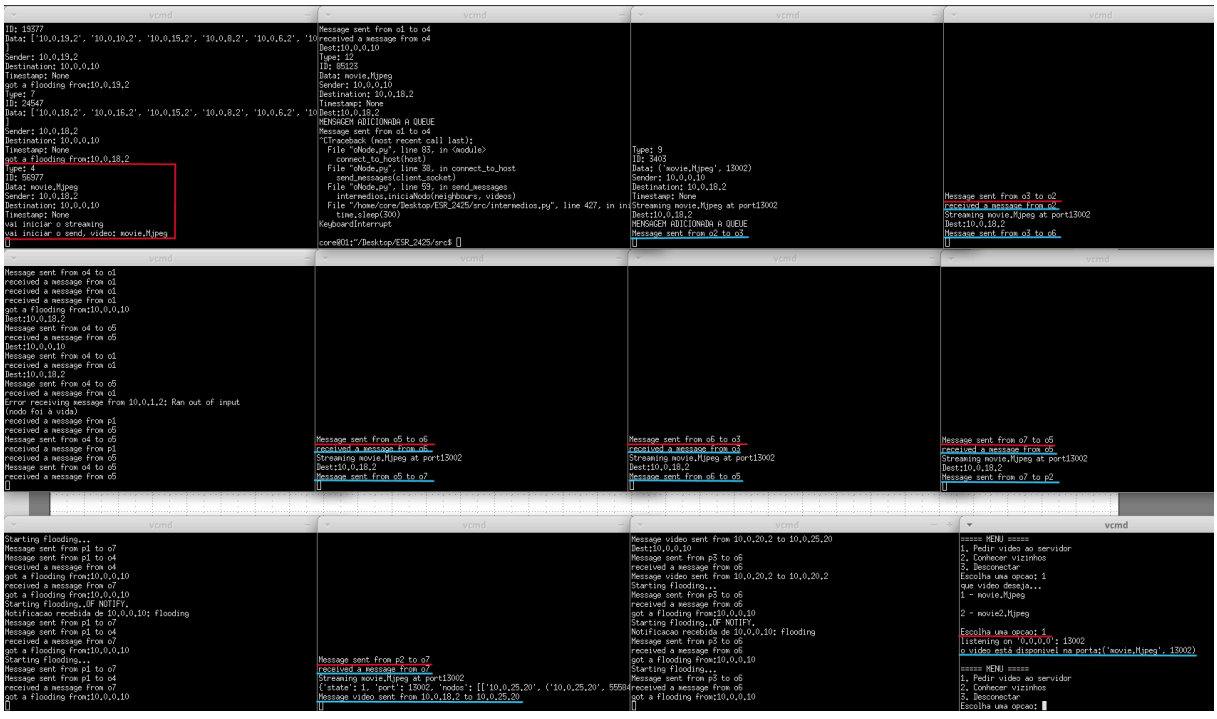

Figura 8: Transmissão de dois videos para um cliente.

Conforme representado na Figura 8, a nova rota estabelecida seguiu um percurso completamente diferente, garantindo a continuidade do serviço. O fluxo do vídeo passou a seguir a rota: *P2 -> O7 -> O5 -> O6 -> O3 -> O2 -> S1*. Este comportamento evidencia a resiliência da arquitetura proposta, que foi capaz de adaptar-se rapidamente às alterações na rede para assegurar a entrega dos conteúdos.

5.2. Teardown de um vídeo

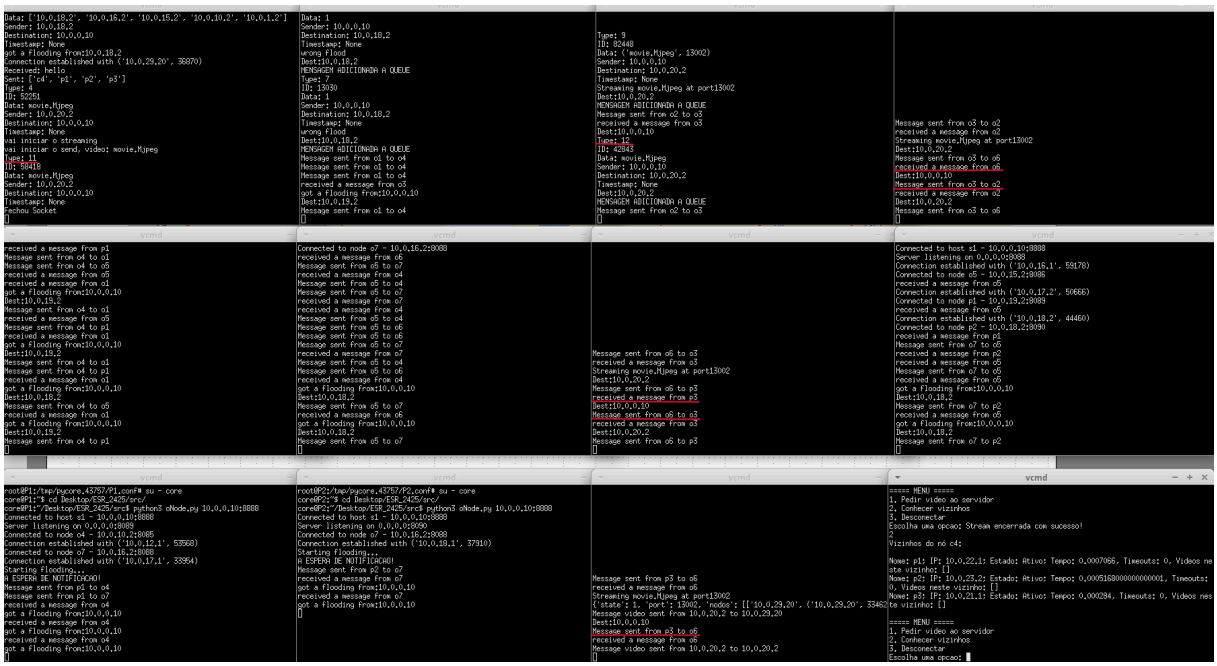

Figura 9: Transmissão de dois videos para um cliente.

Ao selecionar o botão *Teardown*, inicia-se o processo de solicitação para encerrar a transmissão do vídeo. Nesse caso, o pedido seguiu o percurso $P3 \longrightarrow O6 \longrightarrow O3 \longrightarrow O2 \longrightarrow S1$. No terminal do servidor, é possível observar a receção de uma mensagem do tipo 11, indicando o pedido de término da transmissão da *stream*. Em resposta, no caminho de volta, é enviada uma mensagem do tipo 12, representando a confirmação do encerramento da *stream*. Esse fluxo de mensagens assegura o encerramento organizado e eficiente da transmissão.

5.3. Escolha do Melhor PoP

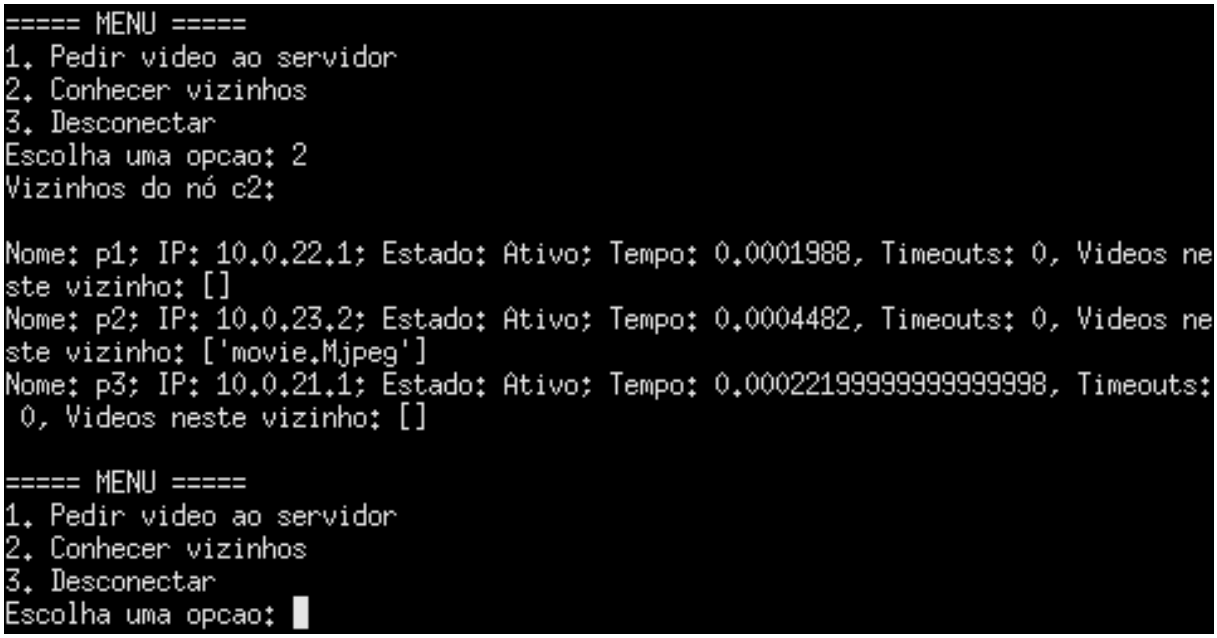

Figura 10: Transmissão de dois videos para um cliente.

Conforme reforçado nos capítulos anteriores, a Figura 10 apresenta uma visão detalhada dos *Points of Presence (PoPs)* disponíveis no sistema. Nesta figura, são exibidos os tempos de resposta associados a cada *PoP*, bem como a lista dos vídeos que estão a ser transmitidos no momento. Essa visualização

facilita a análise da *performance* da rede e auxilia na previsão do *PoP* mais adequado para atender às solicitações dos clientes.

6. Conclusão e Trabalho Futuro

Concluindo, atingimos com sucesso os objetivos propostos, desenvolvendo uma solução eficiente para entrega de conteúdos multimédia em tempo real. Através da arquitetura proposta, foram superados desafios técnicos relacionados à construção da topologia de rede, ao serviço de *streaming* e à gestão de falhas. A escolha estratégica de protocolos, como o RTP para o *streaming* e o UDP e TCP para diferentes níveis de comunicação, assegurou uma solução que equilibra eficiência e . Os testes apresentados comprovaram a robustez e eficiência do sistema, evidenciando a sua capacidade de se adaptar a diferentes condições e oferecer uma experiência otimizada para os utilizadores.

Para trabalho futuro, para além da implementação da etapa complementar de monitorização da rede *overlay*, propõe-se o desenvolvimento de um mecanismo de controlo da rede entre os *Points of Presence (PoPs)* e o servidor. Este controlo permitirá otimizar a distribuição de recursos, reduzindo atrasos e minimizando falhas, assegurando uma experiência de entrega mais robusta e resiliente.