Trabalho Prático Nº2: Serviço Over-the-top para entrega de multimédia

Miguel Luzes[1 [pg55907]], Diogo Barros[1 [a100600]], and Pedro Silva[1 [a100745]]

Universidade do Minho, Braga, Portugal
{pg55907,a100600,a100745}@alunos.uminho.pt

1 Introdução

Com base neste contexto inicial, Este projeto tem como objetivo desenvolver um sistema de entrega de vídeo em tempo real. A solução proposta irá à procura de otimizar a utilização de recursos e garantir uma experiência de alta qualidade para o user. Através do uso do emulador **CORE**, para modelação e teste, o protótipo será baseado numa topologia que integra uma rede CDN e uma rede de acesso, com uma arquitetura overlay aplicacional para a transmissão eficiente dos conteúdos.

2 Arquitetura da Solução

Este projeto tem como objetivo criar um **sistema de streaming em tempo real**, utilizando dois servidores independentes para atender as necessidade de um conjunto de clientes. A topologia proposta é composta por duas principais áreas funcionais: **Access e CDN (Content Delivery Network).** Estas duas áreas estão interligadas por nodos denominados de **Points-of-Presence (PoPs)**.

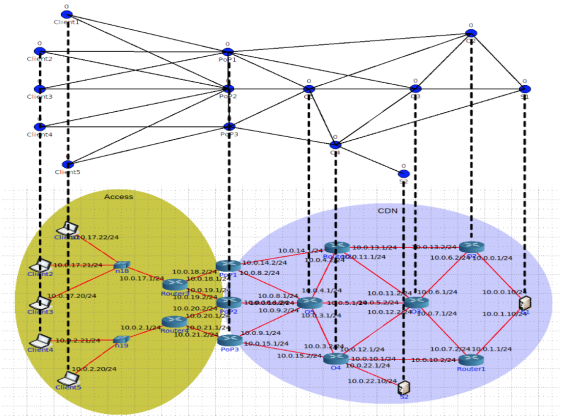

Fig. 1. Topologias Overlay e Underlay do sistema

Os **PoP** são responsáveis por fazer uma ponte entre os clientes e o sistema, distribuindo o conteúdo. São estes que recebem pedidos dos clientes e os encaminham para a rede. Deste modo, é possível reduzir a utilização de recursos da CDN, permitindo a entrega de conteúdos de forma eficiente e contínua. Caso estes nodos já possuam a stream pedida, conseguem tratar do diretamente do fornecimento sem sobrecarregar a rede de distribuição.

Para a implementação do trabalho prático, optámos pela linguagem de programação **Java**, pois ela oferece um bom nível de abstração, tem um grande potencial ao mesmo tempo que é de fácil compreensão, possui uma gama de bibliotecas que facilitam o manuseio de sockets e threads e é de longe a linguagem que mais nos traz conforto ao utilizar.

A estratégia que decidimos implementar para a **construção da rede overlay** passa pela utilização de um **ficheiro de configuração JSON**, como vemos na imagem, que irá conter a informações sobre os **nodos e os seus respectivos vizinhos.**

```
{
    "nodes": [
        {
            "name": "Client1",
            "ip": ["10.0.17.22"],
            "neighbors": ["10.0.18.2", "10.0.19.2"]
        },
        {
            "name": "Client2",
            "ip": ["10.0.17.21"],
            "neighbors": ["10.0.18.2", "10.0.19.2"]
        },
        {
            "name": "Client3",
            "ip": ["10.0.17.20"],
            "neighbors": ["10.0.18.2", "10.0.19.2"]
        },
        {
            "name": "Client4",
            "ip": ["10.0.2.21"],
            "neighbors": ["10.0.20.2", "10.0.21.2"]
        },
```


Fig. 2. Ficheiro de configuração JSON, **bootstrapper**

Quando um nodo, cliente ou servidor inicia a sua execução, este começa um **processo automatizado** de descoberta do ambiente à sua volta. Com o auxílio da função localizada nos **Extras, (List<String> getNeighborsIPs(String nodeIP))**, que recebe o IP do nodo em execução e acede ao ficheiro **bootstrapper**, é possível identificar e listar os IPs de todos os seus vizinhos. Este processo facilita a orientação do nodo, permitindo que ele saiba exatamente com quem está conectado na rede. Este processo é automático e comum ao cliente, nodo e servidor.

Após obter conhecimento dos seus vizinhos, os nodos tratarão de enviar uma mensagem sem contexto aos seus vizinhos para obter informações de qualidade da conexão e para descobrir quais nodos podem estar inativos ou com problemas. Com essas informações, os nodos conseguem organizar os vizinhos por ordem decrescente de qualidade para quando for necessário fazer alguma comunicação. Após este processo, os componentes estão ativos e prontos para receber pedidos.

3 Especificação dos protocolos

No nosso projeto, utilizamos os protocolos de transporte **TCP** e **UDP** para finalidades distintas. Para o protocolo aplicacional do streaming, aproveitamos a implementação do **protocolo RTP (Real-time Transport Protocol)** fornecida pelos docentes. Este é um protocolo que permite o **envio de mensagens** através do protocolo **UDP**, que é **simples e oferece baixa latência**, tornando-o ideal para transmissões em tempo real.

Para a troca de mensagens, optámos pelo **protocolo TCP**, que se revelou uma ferramenta extremamente eficaz, especialmente no que diz respeito à **deteção de falhas**. Além disso proporciona uma gestão eficiente no que toca às informações dos clientes conectados. A combinação deste protocolo com o uso de **threads** conferiu uma estrutura mais organizada ao projeto, permitindo que **cada thread** fosse capaz de gerir de **forma independente um único cliente**, sem comprometer a experiência dos outros utilizadores. A funcionalidade de **retransmissão do TCP**, que permite reencaminhar pacotes de dados que se perdem ou corrompem durante a transmissão, é uma característica essencial e particularmente adequada para o envio de mensagens, onde a integridade e a entrega completa das informações são cruciais.

3.1 Mensagens

O projeto adota um formato de mensagens rigorosamente estruturado, o que otimiza tanto **a sua organização quanto a sua legibilidade.** Cada mensagem é associada a **um identificador único (ID)**, um valor gerado aleatoriamente no **intervalo de 1 a 100.000**, que desempenha um papel crucial, especialmente nos cenários em que as mensagens precisam de ser reencaminhadas.

Tipo de Mensagem	Formato	Definição
`CHECK_VIDEO`	`ID CHECK_VIDEO` nome_video	Verificação da existência de um vídeo
`DISCONNECT`	`ID DISCONNECT`	Pedido de desconexão.
`READY`	`ID READY` nome_video **portaUDP**	Pedido para iniciar a transmissão
`PING`	`ID PING`	Mensagem de ping
`CHECK_STREAM`	`ID CHECK_STREAM` nome_video	Verificação de stream ativa

Fig. 3. Tipos de mensagens existentes no sistema

3.2 Interações

3.2.1 Verificação de existência de um vídeo

Este procedimento inicia-se por um cliente sempre que este deseja aceder a um determinado conteúdo. Por exemplo, suponha que o cliente pretende assistir ao vídeo denominado de **"movie.Mjpeg"**. Para isso, ele envia uma mensagem no formato: `ID CHECK_VIDEO movie.Mjpeg`.

Esta mensagem é encaminhada diretamente para o nodo adjacente na rede. Esse nodo, ao receber a mensagem, realiza uma verificação para determinar se este contém a stream associada ao vídeo solicitado ativa. Caso a stream esteja disponível, o nodo responde imediatamente ao cliente com a **mensagem "True".**

Se o nodo **não possuir a stream do vídeo solicitado**, este reencaminha a mensagem para os seus nodos vizinhos. Durante este reencaminhamento, o ID da mensagem original é **preservado** e desempenha um papel fundamental, pois evita que a mensagem entre num **ciclo de retransmissão infinito**, visto que este projeto, está desenhado para que cada nodo na rede processe a **mensagem apenas uma vez**, mesmo que a receba **múltiplas vezes.**

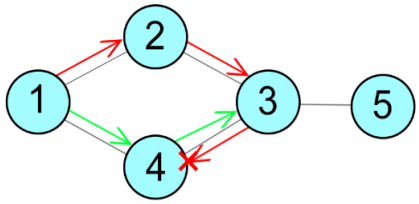

Fig. 4. Representação da propagação de uma mensagem

Por fim, se o pedido for propagado por toda a rede sem que o vídeo solicitado seja **localizado**, o cliente receberá uma mensagem de resposta indicando **"False".** Este mecanismo assegura que a busca pelo conteúdo seja **eficiente**, ao mesmo tempo em que previne **sobrecarga de mensagens redundantes na rede.**

3.2.2 Pedido de vídeo

Depois de se confirmar que o vídeo solicitado está disponível na rede, o cliente inicia um processo de solicitação da transmissão do mesmo. Para isso, o cliente envia ao nodo adjacente uma mensagem no seguinte formato, a pedir pela stream do vídeo: `ID READY <nome_do_vídeo> <porta_UDP>.`

Nesta mensagem, o nome do vídeo identifica o **conteúdo desejado**, e a porta UDP especifica onde o **cliente pretende receber os dados da transmissão.** A porta UDP é gerada aleatoriamente dentro do intervalo de **1024 até 49151**. Para simplificar o processo e considerando que a **probabilidade de conflitos** é extremamente baixa, a porta UDP indicada pelo cliente também será usada pelos nodos para **receber a mesma stream de outros nodos.**

Ao receber o pedido, o nodo verifica se já possui a **stream ativa para o vídeo solicitado.** Caso a stream esteja disponível, ele inicia imediatamente o **envio dos frames do vídeo para o cliente.**

Se o nodo não tiver a stream ativa, este **reencaminha a mensagem** para os seus nodos vizinhos. O reencaminhamento segue uma **métrica de qualidade**, priorizando caminhos que garantam um **melhor desempenho e menor latência para o cliente.** Este processo de reencaminhamento continua até que um nodo com a stream ativa seja encontrado ou, em última instância, até que a mensagem chegue a um dos servidores.

Este mecanismo assegura que o vídeo solicitado seja entregue ao cliente de forma eficiente, aproveitando a estrutura distribuída da rede enquanto **minimiza atrasos e redundâncias.**

3.2.3 Pedido de desconexão

O pedido de desconexão é utilizado para **interromper uma transmissão de vídeo.** Quando um cliente decide parar de assistir ao conteúdo, ele envia uma mensagem de desconexão no seguinte formato: `ID DISCONNECT.`

Ao receber essa mensagem, o nodo adjacente interrompe imediatamente o **envio de dados para o cliente.** Além disso, o nodo **decrementa o contador** que regista o número de clientes (viewers) conectados à stream que está a ser transmitida.

Caso o número de viewers atinja **zero**, ou seja, não haja mais clientes conectados à transmissão, o nodo também enviará uma mensagem de desconexão ao nodo fornecedor da stream. Este procedimento garante a **minimização do número de transmissões ativas**, libertando os recursos da rede que poderão ser utilizados para outros propósitos.

3.2.4 Ping

A **monitorização ativa** é uma das funcionalidades fundamentais do sistema, sendo essencial para garantir a **estabilidade e o desempenho ideal** das transmissões. Todos os nodos da rede verificam periodicamente o estado das conexões com os seus vizinhos, assegurando que as **melhores condições de transmissão** sejam mantidas para os clientes.

Logo após a inicialização, cada um dos nodo da rede envia mensagens de ping no seguinte formato: `ID PING.`

Estas mensagens são enviadas a todos os nodos vizinhos, para facilitar a **criação de métricas de qualidade** das conexões e identificar vizinhos que possam estar **inativos.**

Para além disso, os nodos também avaliam o **estado das conexões** sempre que recebem uma **solicitação de transmissão (READY)**. Este procedimento permite identificar a **entrada de ou a saída de nodos** existentes na rede.

3.2.5 Verificação de stream ligada

Para complementar as métricas de qualidade e a eficiência da rede, além do envio de pings, os nodos realizam umas **verificações adicionais** junto com os seus vizinhos para determinar se estes possuem streams ativas. Esta funcionalidade tem como objetivo estabelecer prioridades na escolha dos nodos mais adequados para o reencaminhamento de pedidos de transmissão.

No contexto de seleção de caminhos eficientes, cada nodo, ao avaliar qual o vizinho que possui as melhores condições para atender as suas necessidades, envia uma mensagem no seguinte formato: `ID CHECK_STREAM <nome_do_vídeo>.`

A resposta à mensagem, pode ser **TRUE**, se o nodo possui uma stream ativa ou **FALSE**, se o nodo se encontra com a stream inativa.

Com base nas respostas recebidas, os nodos conseguem priorizar aqueles que já possuem a stream ativa, reduzindo a necessidade de iniciar novas transmissões. Este mecanismo não apenas minimiza a utilização de recursos da rede, mas também contribui para uma redução significativa na latência e na sobrecarga de mensagens.

4 Implementação

4.1 Construção da topologia underlay e planeamento da topologia overlay

Numa primeira etapa, foi desenvolvida a topologia **underlay** com o objetivo de criar alguma **redundância** e ao mesmo tempo, ter a possibilidade de **criar caminhos alternativos para transmissão.** Este equilíbrio é essencial para garantir a **resiliência da rede** e permitir **maior flexibilidade no encaminhamento de dados.**

Para auxiliar com a construção da topologia, o sistema utiliza um ficheiro em formato JSON que contém todas as informações necessárias sobre os **vizinhos de cada nodo.** No momento da inicialização, cada nodo acede a esse ficheiro, onde identifica os **seus vizinhos** e armazena essas informações na sua base de dados.

Após a obtenção dos dados sobre os vizinhos, os nodos enviam mensagens de ping para avaliar o **estado das conexões** e determinar quais os vizinhos que apresentam as **melhores condições para estabelecer interações.**

A **entrada e saída de vizinhos** na rede é detectada de forma automática por meio dos pings realizados pelos nodos. Por exemplo, caso um cliente solicite uma stream a um nodo que tenha detectado um **vizinho inativo,** os novos pings enviados em resposta à solicitação do cliente podem identificar a **reentrada desse vizinho** na rede, atualizando assim a topologia de forma dinâmica.

4.2 Cliente

A aplicação **Cliente** é utilizada pelos utilizadores, para que estes possam consumir os conteúdos fornecidos pelo sistema. Após a iniciação da mesma, a aplicação obtém os seus **Points of Presence (PoPs)** e estabelece uma **conexão via TCP** com o melhor PoP disponível (porta 33350). Em seguida, a mesma questiona o utilizador sobre qual é o vídeo que este deseja assistir.

O cliente realiza um pedido de verificação específico, que verifica se o vídeo solicitado está disponível na rede. Caso receba uma **resposta negativa**, esta informa o utilizador **(via terminal)** e encerra a aplicação. Se a resposta for **positiva**, a interface é aberta, e o cliente prepara-se para **receber a transmissão.** Devido a uma pequena limitação na **conversão das cores do vídeo original**, as frames necessitam de passar por um conversor que trata de **ajustar as cores** para que o vídeo seja visualizado corretamente. No entanto, esta etapa não se aplica a todos os outros vídeos.

Durante a transmissão, o cliente emite **pings periódicos aos seus Pops**, e se identificar com sucesso, que o PoP atual deixou de ser o **mais eficiente**, vai efetuar uma troca para um outro PoP, em **tempo real e sem interrupções na transmissão**, que demonstrou ser **mais eficiente.**

A aplicação **Cliente** suporta também funcionalidades como **pausar e retomar o vídeo.** Quando o vídeo é retomado, mesmo sendo em tempo real, ele continua no **ponto atual do servidor.** Além disso, o botão de **"stop"** (ou o encerramento da janela) envia um aviso ao PoP para **interromper a transmissão** e só depois é que **encerra a aplicação.**

4.3 Serviço de Streaming

Decidimos implementar uma estratégia de streaming, mais adaptada e otimizada para a nossa arquitetura.

O processo de inicialização de uma stream é feito pela **thread associada ao cliente** que a solicita. Esta thread verifica na base de dados, se a **stream do vídeo pedido** já está ativa, caso não esteja, a thread trata de **criar a stream** desse vídeo.

O streaming em cada nodo é da responsabilidade de **duas threads.** Em cada nodo intermédio, o último frame da stream e o seu tamanho são **armazenados em cache.** Para além disso, é armazenado um **contador de nodos conectados.** A **primeira thread** é responsável por manter o **frame atualizado**, em função do servidor. Sempre que houver **clientes conectados**, estes acedem a essa cache para puderem obter o **frame mais recente.** Isto significa que todos os nodos vão ver o mesmo frame, satisfazendo assim o requisito de **stream em tempo real.**

Quando todos os clientes deixarem de assistir essa stream, o contador de nodos conectados será nulo, o que faz com que a **thread principal** seja encerrada. A **segunda thread**, por sua vez, aguarda pela **finalização da primeira**, para que depois possa remover a stream da base de dados e notificar o nodo que a forneceu, que este não necessita de transmitir mais frames da mesma.

O nosso sistema de streaming contém duas variações, dependendo se ocorre num **nodo intermédio ou no servidor.** Esta diferença está na maneira como o nodo obtém o frame. No **servidor**, a thread principal do streaming acede à memória onde os **vídeos estão previamente carregados.** No **nodo**, o frame é obtido por transmissão de um **nodo fornecedor do frame.** Fora isto, a lógica do streaming é similar para ambos os casos.

4.4 Construção das Árvores de Distribuição

A **construção da árvore de distribuição** inicia-se no cliente, após a confirmação de que o vídeo solicitado existe. O cliente envia uma mensagem **"READY"** para o seu **Point of Presence (PoP).** Este nodo verifica se a stream solicitada já está **ativa**, caso não esteja, este trata de **reencaminhar o pedido.** O reencaminhamento é realizado com base nos seguintes pontos:

- **Verificação de vizinhos:** o nodo envia pings para os vizinhos a fim de identificar os melhores caminhos e descartar os que possam estar inativos.
- **Disponibilidade da stream:** o nodo questiona os vizinhos se já possuem a stream ativa, atribuindo prioridade a esses caminhos. Contudo, essa prioridade é condicionada pela qualidade da conexão. Se um vizinho tiver a stream ligada, mas com uma conexão de baixa qualidade, o nodo optará por um caminho alternativo mais eficiente.

Durante o processo de reencaminhamento de pedidos, os nodos também realizam uma **monitorização ativa** para garantir a qualidade da transmissão.

Este processo é **iterativo** e funciona no estilo **DFS (Depth First Search).** Cada nodo questiona todos os seus vizinhos, sobre a **disponibilidade da stream**, antes de responder ao nodo que efetuou o **pedido inicial.** Da mesma forma, cada nodo que recebe o pedido, encaminha também o pedido a todos os **seus vizinhos.** A figura a seguir ajuda a ilustrar este processo:

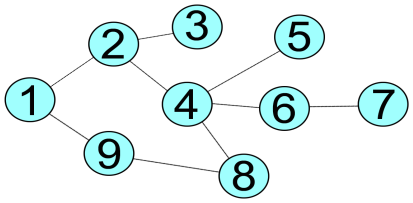

Fig. 5. Representação da ordem de visita dos nodos

É garantido afirmar que o caminho encontrado é o melhor porque cada nodo prioriza, na sua pesquisa, sempre o melhor nodo com base nas métricas definidas.

Se todos os caminhos apresentarem uma **qualidade demasiado baixa** (com tempos de resposta superiores ao limite estipulado de **100ms**), então a transmissão **não terá condições** para ser iniciada, e o cliente será **notificado desta indisponibilidade.**

Quando o pedido **chegar ao servidor**, este responderá positivamente e irá preparar a inicialização da stream. Cada nodo irá receber a **resposta positiva e irá realizar o mesmo procedimento**, até que o caminho esteja totalmente configurado até **chegar ao cliente**, garantindo que este será o **caminho ideal.**

Caso, durante a pesquisa, seja encontrado um nodo com a **stream já ativa para um outro cliente**, este nodo responde positivamente e preparar-se para **enviar frames**, estabelecendo assim um novo caminho até ao **novo cliente.**

Este processo é **dinâmico**, ou seja, os nodos não armazenam nenhum caminho. Estes apenas têm conhecimento sobre **a origem e o destino da stream**, devido às conexões TCP abertas, pela qual podem enviar pedidos de desconexão. Caso um cliente **pare de assistir um vídeo** e volte a solicitar o mesmo vídeo mais à frente, o caminho poderá ser **diferente.** Isto ocorre por causa das **alterções das condições de rede**, garantindo assim que a árvore é sempre ótima no **momento do pedido.**

4.5 Extras

Para além das funcionalidades dos programas principais que constituem o nosso sistema, desenvolvemos alguns **programas auxiliares**, recorrendo à linguagem **Python**, que nos ajudaram bastante durante o desenvolvimento do projeto.

4.5.1 Visualizador da rede overlay

Este programa permite visualizar a rede **Overlay** de uma outra perspetiva. Em vez de se utilizar mensagens imprimidas no terminal, que podem ser difíceis de interpretar, desenvolvemos um programa que é capaz de **desenhar a rede overlay graficamente.** Assim, torna-se muito mais fácil visualizar as árvores de distribuição, ver quantos nodos estão a usufruir de uma transmissão, e em geral, torna-se mais fácil de encontrar problemas no nosso projeto.

4.5.2 Conversor de vídeos

Inicialmente, o código fornecido pelos docentes obtinha frames de um vídeo, lendo o **tamanho de cada frame** e usando esse tamanho para determinar a **quantidade de dados a serem lidos** para obter o frame. No entanto, após tentarmos vários conversores online de `MP4` para `MJPEG`, descobrimos que os vídeos produzidos não seguiam este formato, então decidimos criar um **pequeno conversor**, que adapta estes vídeos `MJPEG` convencionais para conseguirem ser lidos pelo sistema.

Este programa disfruta do uso da biblioteca `CV2` da linguagem **Python**, que convenientemente consegue ler o vídeo **frame a frame.** O programa cria um **novo vídeo**, calcula o **tamanho de cada frame** e coloca o **tamanho calculado** seguido do frame que leu. Este processo é feito para todos os frames.

5 Testes e Resultados

Realizamos vários testes para avaliar o **desempenho da solução implementada** e que abrangem diversos cenários.

5.1 Stream para mais que um cliente

Numa primeira instância, fizemos um teste básico de stream com **dois nodos intermédios.** Foram abertos o `PoP1` e o `O2`. Com os nodos em execução abrimos dois clientes `(Client1 e Client2)` que solicitaram o **mesmo vídeo.**

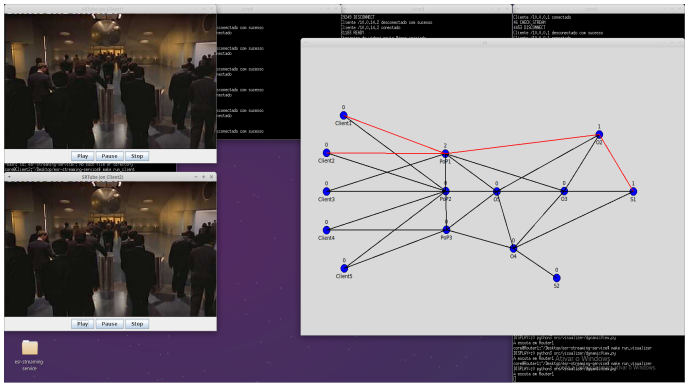

Fig. 6. Transmissão para dois clientes

Conseguimos verificar que o `PoP1` tem **dois clientes conectados**, e que o `O2` e o servidor `S1` têm apenas um. Isto demonstra que o **fluxo único** de `S1` para o `PoP1` é depois replicado para os **dois clientes.**

Em baixo, podemos visualizar as interações entre os nodos:

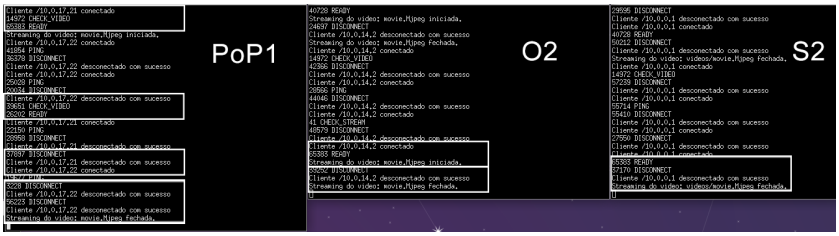

Fig. 7. Interações no teste 1

O `PoP1` recebeu **dois pedidos "READY"**, sendo que o primeiro foi reencaminhado para o `O2` e para o `S1`. O segundo pedido obteve uma **resposta positiva** do `PoP1`, pelo que não teve necessidade de ser encaminhado. Em seguida, os dois clientes enviam pedidos de `"DISCONNECT"`, sendo que o segundo desencadeia pedidos de `"DISCONNECT"` até ao **servidor.**

5.2 Teste com dois servidores

Neste teste, utilizamos os **dois servidores ao mesmo tempo.** Para funcionar da melhor forma, todos os nodos da parte central da topologia foram **desativados** e colocamos **um atraso no link** entre o `O4` e o `S1`, para garantir que as duas árvores não se juntam. Neste caso, como as streams são provenientes de dois servidores diferentes, os vídeos **não estão sincronizados.**

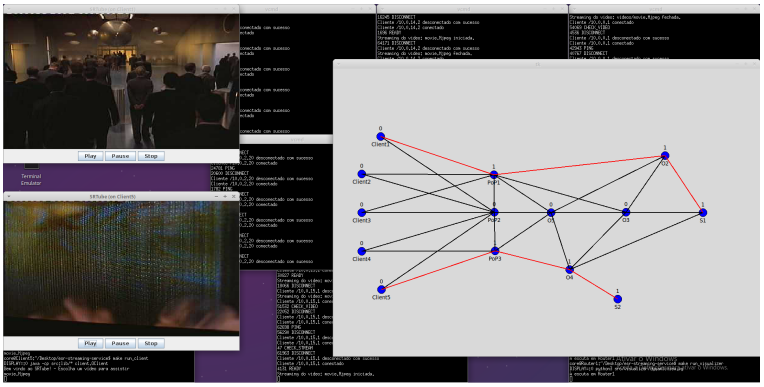

Fig. 8. Teste com dois servidores (delay adicionado)

5.2.1 Remoção do atraso entre O4 e S1

Este é um caso extra, para mostrar a **reutilização da stream** do `S1`. Como o `O4` é vizinho dos dois servidores, ao remover o **atraso entre ele** e o `S1`, a stream do `S1` vai ser escolhida. Deste modo, a stream do `S1` é aproveitada e poupam-se recursos do `S2`.

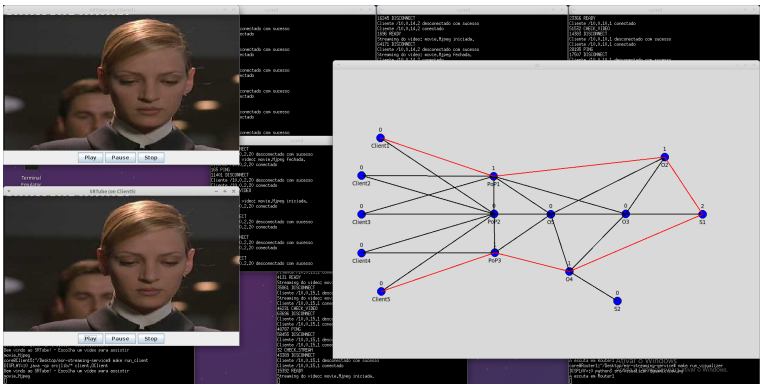

Fig. 9. Teste com dois servidores (sem delay)

5.3 Caminhos alternativos em caso de atrasos

Este teste tem como objetivo mostrar a **adaptação dos caminhos**, na ocorrência de atrasos em certos links. Para isso, criamos um cenário com vários atrasos para testar a capacidade do sistema em procurar **um caminho alternativo.**

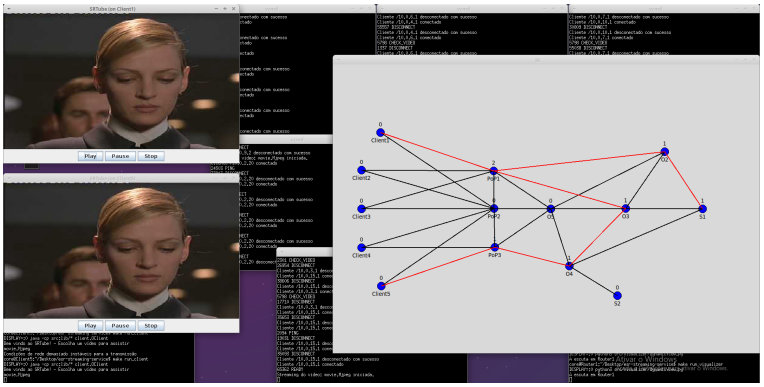

Fig. 10. Teste extremo de caminhos alternativos

Este caso é bastante extremo. Foram adicionados atrasos nos links, `PoP1-O5`, `O5-O3`, `O5-O2` e `O4-S1`, forçando o **pathfinder**, a recorrer a um caminho alternativo e bastante atípico. Este teste mostra a capacidade de adaptação do sistema a condições adversas.

6 Conclusão e Trabalhos Futuros

Neste trabalho, conseguimos implementar um sistema funcional e que acreditamos estar de acordo com a maior parte os requisitos propostos. A distribuição das streams pelos nodos permite fornecer conteúdos a vários clientes **minimizando a quantidade de dados transmitidos.** O nosso sistema consegue adaptar-se às condições da rede, procurando sempre proporcionar **a melhor qualidade possível de transmissão**.

Por outro lado existem alguns pontos que poderiam ser melhorados:

- **Monitorização:** as métricas poderiam disfrutar de outras estatísticas como por exemplo, um sistema de probing para determinar a distância de cada nodo ao servidor.
- **Melhor adaptação da rede a perdas:** melhorar a forma como os nodos reagem no caso de um nodo transmissor tenha problemas (procura de caminhos alternativos).
- **Ajuste da qualidade de vídeo:** implementação de uma funcionalidade que ajuste automaticamente a qualidade do vídeo com base, na qualidade da conexão, permitindo consumir conteúdos em cenários menos favoráveis.

Concluimos este trabalho refletindo sobre o que foi feito e apesar de haver pontos que podiam ser melhorados, estamos satisfeitos com o resultado final, tendo em conta os requisitos propostos.