Engenharia de Serviços em Rede - Serviço Over the Top para entrega de multimédia - TP2

Francisca Lemos, Nuno Costa, and João Neves

[1] Universidade de Minho
[2] Departamento de Informática, Universidade do Minho, Braga, Portugal
{pg50497,pg52693, pg52698}@alunos.uminho.pt

1 Introdução

Este trabalho tem como objetivo criar um protótipo de um serviço OTT, visando a eficiência e a otimização de recursos para proporcionar uma melhor qualidade de experiência ao utilizador.
A ideia central é desenvolver uma rede overlay própria, fundamentada em protocolos de transporte (TCP ou UDP) e/ou aplicacionais (HTTP) da Internet, com o intuito de melhorar a entrega de conteúdo. Para cumprir com o objetivo, foi construída uma rede overlay onde os nós atuam como intermediários no envio dos dados, tendo em conta a latência sentida entre dois nós, para evitar atrasos significativos. Para isso a estratégia usada foi descobrir os caminhos menos custosos desde o servidor até o cliente definindo como métrica a latência.

No decorrer deste relatório serão explicados todas as decisões tomadas ao longo do desenvolvimento do trabalho prático, tal como a arquitetura da solução (**Secção 2**); especificar os protocolos e as mensagens (**Secção 3**); apresentar a nossa implementação (**Secção 4**); os limites que a nossa solução apresenta (**Secção 5**); e por último, mostrar os testes efetuados (**Secção 6**)

2 Arquitetura da solução

Começamos por definir a linguagem de programação utilizada para desenvolver o projeto e esta foi **Java**, por ser confortável para todos os elementos do grupo. Ao longo das diferentes etapas foram construídos vários cenários de testes que serão apresentados na respetiva secção.
Relativamente aos protocolos de transportes, utilizou-se o protocolo TCP para estabelecer conexões entre os Clientes e os Servidores e para controlar o envio de conteúdo. Além disso, utilizou-se o protocolo UDP para o envio de conteúdo para streaming.

Decidimos utilizar a estratégia do ficheiro YAML escrito manualmente que contem todos os vizinhos de um nó (seja Cliente, Servidor ou RP) e os respetivos endereços IP's. Primeiramente, implementamos uma classe que realiza parse do ficheiro e armazena numa estrutura de dados as informações. Posteriormente, o RP iniciou o processo de implementação de um mecanismo de troca de mensagens entre os nós, incluindo informações sobre latências. Para a solicitação de um

video ao Servidor, implementamos o algoritmo de Dijkstra, que nos fornece o caminho de custo minimo, onde os pesos são a latencia sentida entre os nós. Dessa forma, o RP utiliza o resultado do algoritmo para efetuar a requisição ao servidor, otimizando a escolha do caminho com base nas latências observadas.

3 Especificação dos protocolos

3.1 Formato das mensagens protocolares

Foi implementado uma classe *Messages* que contém as mensagens protocolares trocadas entre os diversos elementos da nossa topologia.
As mensagens protocolares entre o RP e o Cliente são as seguintes:

- **request:** Serve para o Cliente solicitar ao nó mais próximo o conteúdo que pretende ver;
- **received:** O RP confirma ao Cliente que recebeu o pedido do vídeo;
- **file:** O RP avisa o Cliente que tem o vídeo solicitado para, posteriormente, o enviar;
- **accepted:** O Cliente aceita a conexão com o RP;
- **no_video:** O Cliente quer parar de ver o vídeo;

O RP utiliza essas mensagens para requisitar um vídeo específico ao Servidor, que por sua vez responde a essa solicitação, estabelecendo assim um canal de comunicação:

- **request_stream:** O RP pede o vídeo ao Servidor;
- **start_stream:** O Servidor avisa que vai começar a enviar o vídeo ao RP;
- **start_ack:** O RP recebe a confirmação do envio do vídeo;
- **send_stream:** É enviado a cada frame do vídeo;
- **end_stream:** O RP vai fechar a conexão com o Servidor;

4 Implementação

4.1 Detalhes, parâmetros, bibliotecas de funções,etc

Como a estratégia utilizada foi a de um ficheiro, começamos por analisar o ficheiro através da classe *Leitor*. Nesta etapa, ocorre o parse do ficheiro, e em seguida, os vizinhos são armazenados numa HashMap.

No que toca ao serviço de streaming, implementou-se um *Cliente* e um *Servidor* simples com base do código do livro de apoio facultado pela equipa docente. O servidor lê o vídeo de um ficheiro, enviando-o por pacotes numerados ao cliente, onde é reproduzido numa nova janela. Nesta janela existem 4 botões: Setup, Play, Pause e Teardown. Quando o botão Setup é pressionado, é criado um *DatagramSocket* para posteriormente receber o conteudo do vídeo, e altera o estado para **INIT**. Caso o Play seja pressionado, inicializa o timer e, consequentemente,

o vídeo é reproduzido (mudando o estado para **READY**). O botão Teardown serve encerrar o vídeo. Quando o pacote *RTP* chega ao cliente, a imagem do vídeo é transmitida numa ordem especifica, consoante o número de sequência de cada frame. Se o número de sequência atual for maior que o número de frame, o pacote é descartado. O *Servidor* responde a cada pedido efetuado pelo *Cliente*.

O RP inicia a configuração da topologia criando duas threads: uma para a classe *Functional*, responsável pela interpretação de mensagens, e a outra para a classe *Latency*, encarregada de gerenciar as informações de latência. Após isso, verifica as latências de todas as conexões para conseguir criar caminhos de custo minimo. A latência indica o tempo de resposta entre os diferentes pontos da rede, calculado como a diferença entre o momento em que o pedido é recebido e o momento em que é enviado, utilizando a função *Calendar.getInstance().getTime().getTime()*.

Na construção das rotas para o envio do conteúdo utilizamos o algoritmo de Dijkstra. Este algoritmo é utilizado para determinar o caminho de custo mínimo, considerando como "pesos"a latência entre os nós, permitindo a escolha do caminho mais eficiente em termos de tempo. Na classe *Maintence*, a cada intervalo de 5 minutos, o RP verifica se os nós ainda estão ativos. Caso não obtenha resposta de algum deles é atribuída uma latência elevada para as conexões relacionadas com ele.

5 Limites da Solução

A nossa solução final apresenta algumas limitações. Em primeiro lugar, a composição da topologia permanece igual, ou seja, não é possível que os nós da árvore fiquem indisponíveis. Isso faz com que o nosso projeto não se assemelhe ao mundo real, onde a perdas de nós é uma ocorrência comum. Além disso, a ausência da implementação da estratégia do **bootstrapper** é outra lacuna. Faz com que reduza significamente o dinamismo da solução, pois toda a topologia tem de ser escrita manualmente, o que torna o processo de adição de novos nós uma tarefa complexa e propensa a erros. Com um bootstrapper, este novo nó poderia entrar na topologia, identificando-se diretamente ao RP. Estas melhorias aumentariam a escalabilidade do nosso projeto.

6 Testes e resultados

Como cenário de teste, construímos a rede overlay ilustrada na Figura 1, com um Servidor, 2 nodos intermediários e 2 Clientes.

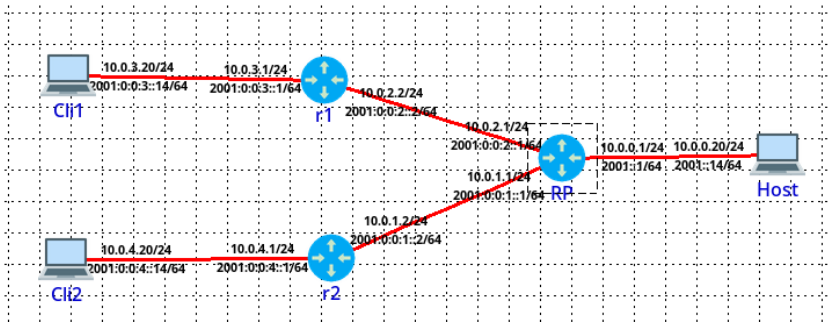

Figura 1. Rede overlay para cenário de teste

O seguinte teste mostra os 2 clientes a consumir o vídeo em simultaneo.

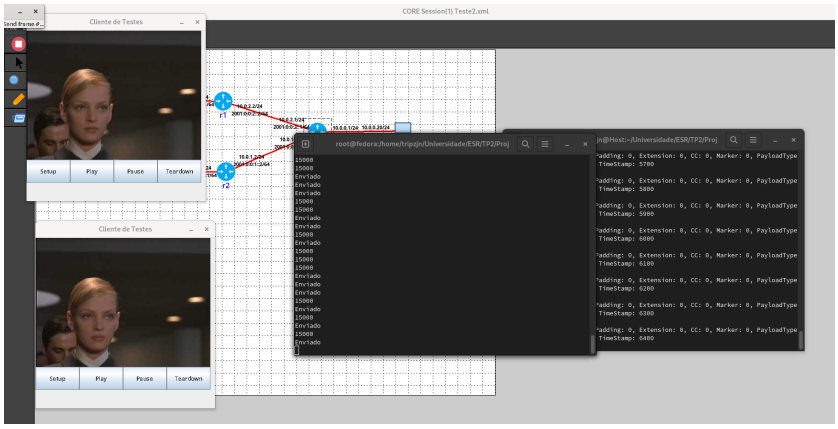

Figura 2. Exemplo de teste bem sucedido

7 Conclusões e trabalho futuro

Este trabalho ajudou-nos a consolidar conceitos lecionados durante o semestre relativos a serviços de streaming de vídeos e protocolos.

Entretanto, apesar da nossa solução cumprir o objetivo principal, percebemos algumas limitações importantes. Uma delas é a topologia não suportar a adição e perda de nós, tornando o sistema mais flexível e adaptável a mudanças no funcionamento da rede em condições reais. Além disso, a ausência de um bootstrapper faz com que todo o processo de construção da topologia tenha de ser feito manualmente, dificultando a expansão e tornando o sistema mais vulnerável a erros.

Como trabalho futuro, pretendemos resolver as lacunas referidas anteriormente. Em suma, este projeto teve uma enorme importância para a aplicação de conceitos teóricos e práticos lecionados na unidade curricular e como grupo consideramos o balanço deste projeto positivo e que os requistos propostos foram cumpridos.