Stream Audio Processor

1.º Pedro Duarte
Engenharia Eletrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201905050@edu.fe.up.pt

2.º Ruben Santos
Engenharia Eletrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201905773@edu.fe.up.pt

***Resumo*—O presente documento descreve o trabalho realizado no desenvolvimento de um processador de áudio para *streaming*, no âmbito da unidade curricular Áudio Computacional. O principal objetivo deste projeto compreendia conceber e implementar um sistema de *streaming on demand*. Na fase inicial de conceção do sistema, exploraram-se diversas plataformas de *streaming*, nomeadamente os protocolos utilizados, características de *layout* e funcionalidades. Já na fase de implementação, a linguagem de programação Python foi empregue para desenvolver a aplicação, a qual foi nomeada de "pobrefy", tendo-se utilizado a *framework* de multimédia FFmpeg para a codificação do áudio e o servidor Nginx como servidor de *streaming*. O sistema final, embora apresente algumas limitações em relação às amplas funcionalidades oferecidas pelos serviços de *streaming* existentes no mercado, permite realizar a premissa básica deste tipo de sistemas: selecionar uma música e reproduzi-la *on demand*, sem a necessidade do cliente ter a música presente no seu dispositivo. Além disso, a aplicação oferece um conjunto de funcionalidades básicas de manipulação da reprodução, como parar, ajustar o volume e avançar ou retroceder para um dado instante.**

***Index Terms*—pobrefy, *streaming*, *on demand*, MP3, HTTP Live Streaming, Advanced Audio Coding, Nginx, FFmpeg, VLC, Pyhton, PYQT6.**

I. Introdução

Nos últimos anos, o domínio do *streaming* de áudio tem evoluído consideravelmente, com a capacidade de transmitir áudio instantaneamente a tornar-se um elemento indispensável no universo da música digital. Plataformas populares, como Spotify e Deezer, lideram esse movimento, oferecendo uma gama diversificada de funcionalidades para aprimorar a experiência do utilizador.

Neste contexto, surge o projeto "pobrefy", concebido como um sistema de *streaming on demand*, explorando protocolos, codecs e servidores para proporcionar uma solução eficiente e acessível para o consumo de áudio em tempo real. Este documento oferece uma análise aprofundada das fases cruciais do projeto, desde a conceção à implementação, destacando as decisões estratégicas tomadas durante esse percurso. São também abordadas as funcionalidades e as considerações que nortearam o processo de design. Por último, são apontadas possíveis melhorias e expansões, visando a contínua evolução do "pobrefy".

II. Conceptualização do sistema

Na fase inicial deste projeto, foi conduzida uma pesquisa abrangente acerca do estado da arte nos serviços de *streaming* modernos, como Spotify e Deezer, com o objetivo de analisar as funcionalidades disponíveis para os utilizadores, os protocolos de *streaming* empregues e a arquitetura do *backend* desses serviços.

A. Protocolos de Streaming

Um protocolo de *streaming* é fundamental para definir as regras e processos pelos quais um conteúdo multimédia é entregue aos utilizador. Existem diversos protocolos de *streaming*, cada um com características distintas, tornando-os mais adequados para finalidades específicas. De entre os vários protocolos existentes, foram explorados três de forma mais detalhada:

- **HTTP Live Streaming (HLS)**: é um protocolo de streaming desenvolvido pela APPLE Inc., utilizado para o *streaming* de multimédia *on demand* ou ao vivo, que divide uma música ou vídeo em diferentes segmentos com a duração de alguns segundos. A informação sobre esses segmentos é fornecida através de uma *playlist* M3U8, que são ficheiros que contêm URLs para os diferentes segmentos. Para além disso, este protocolo utiliza HTTP para a entrega dos segmentos, o que possibilita utilizar servidores web convencionais. A principal característica deste protocolo é a capacidade de adaptação às condições de rede do utilizador, conhecida como Adaptive Bit Rate (ABR), permitindo a troca entre diferentes taxas de bits para uma experiência de audição ou visualização mais suave e com menor tempo de buffering.
- **MPEG-DASH**: este protocolo foi desenvolvido pela MPEG e funciona de forma similar ao HLS, dado que divide o conteúdo multimédia em *chunks* ou segmentos de alguns segundos. O protocolo cria um ficheiro MPD que contém informação sobre a localização dos segmentos, *bitrates* e outros detalhes importantes. Para além disso, utiliza o protocolo HTTP para a entrega de segmentos e também é compatível com ABR, ou seja, é possível ocorrer uma troca entre *bitrates* dependendo da situação da rede. Contudo, este protocolo apresenta um maior versatilidade do que o HLS, dado que pode ser utilizado por um maior número de sistemas. Para além disso, o DASH suporta recursos avançados como o DRM e transmissões multimédia simultâneas.
- **Real-time Transport Protocol (RTP)**: é um protocolo de *streaming*, normalmente utilizado para transmitir áudio e

vídeo ao vivo, sendo possível verificar a sua utilização em conferências por vídeo e *Voice over IP* (VoIP). É comum utilizar este protocolo juntamente com o protocolo Real-time Transport Control Protocol (RTCP), que permite obter *feedback* sobre a qualidade dos dados transmitidos. Para além disso, o RTP recorre ao protocolo da camada de transporte UDP para realizar o transporte dos dados.

Com base nisto, escolheu-se o protocolo de streaming HLS, como protocolo a ser utilizado neste trabalho, uma vez que o protocolo RTP é mais adequado para a transmissão de música ao vivo e o protocolo DASH, apesar de ser mais versátil e possuir um maior número de funcionalidades, é mais complexo de ser implementado do que o HLS; para este projeto, só é necessário utilizar as funcionalidades básicas destes protocolos.

Para além da escolha do protocolo, foi necessário selecionar ferramentas que permitissem codificar, no servidor, e descodificar, no cliente, o conteúdo que é transmitido. Como codificador selecionou-se a *framework* de multimédia FFmpeg, uma *framework open-source* que permite, entre várias funcionalidades, codificar diferentes codecs para vários protocolos de *streaming*, entre eles HLS. Em termos de descodificador, decidiu-se usar uma biblioteca da linguagem de programação Python, Python-VLC, que é um *wrapper* do reprodutor de multimédia *open-source* VLC, o que permite incorporar as funcionalidades do VLC, principalmente o descodificador e o reprodutor multimédia, possibilitando descodificar e reproduzir os segmentos recebidos no cliente.

B. Codecs

A maior parte das músicas disponibilizadas online são comprimidas com recurso ao codec MP3. No entanto, surge a necessidade de as converter para Advanced Audio Codec (AAC) devido à escolha do protocolo HLS para entrega do conteúdo, protocolo este otimizado para operar com o codec em questão. O AAC é reconhecido como um padrão na transmissão de áudio via *streaming*, oferecendo uma melhor eficiência de compressão e qualidade de áudio em comparação com o MP3, especialmente em taxas de bits inferiores, como os 128 Kbits/s a que o áudio é codificado no "pobrefy". Por último, o AAC é amplamente suportado em dispositivos e navegadores.

C. Servidor

Na seleção do servidor, a decisão de adotar o protocolo HLS como padrão para o serviço de *streaming* teve implicações significativas. Essa escolha estratégica eliminou a necessidade de desenvolver um servidor dedicado, uma vez que o HLS permite a recuperação de segmentos específicos diretamente de um servidor web convencional. A capacidade do protocolo em utilizar uma infraestrutura web comum para armazenar e distribuir diversas versões dos dados de média simplificou substancialmente a implementação, proporcionando eficiência operacional e facilidade de manutenção. Deste modo, a escolha foi reduzida a dois servidores:

- **Apache HTTP Server**: o Apache é um servidor *open-source*, compatível com o protocolo HTTP e que pode ser configurado para servir conteúdo de *streaming*, tendo, contudo, suporte limitado para vários protocolos incluindo HLS. A principal característica deste servidor é a sua arquitetura e a forma como lida com múltiplas conexões, uma vez que o Apache é um servidor baseado em *threads*. A cada nova conexão é criada uma *thread* para lidar com o pedido, o que pode levar a um aumento no consumo dos recursos computacionais em situações de tráfego intenso.
- **Nginx**: o Nginx é um servidor *open-source*, compatível com o protocolo HTTP e que pode ser utilizado como *proxy* reverso. Um servidor Nginx também pode ser configurado para servir conteúdo de *streaming*, apresentando um suporte alargado para vários protocolos. Por sua vez, este servidor tem uma arquitetura assíncrona e orientada a eventos, o que o torna mais eficiente em situações de tráfego intenso, dado o consumo reduzido de recursos computacionais.

Devido ao suporte mais robusto para conteúdo multimédia e *streaming*, juntamente com a eficiência em situações de tráfego intenso e menor consumo de recursos computacionais, o Nginx foi escolhido como o servidor a ser utilizado na implementação final.

D. Aplicação

As aplicações de *streaming* de música que são líderes de mercado dão aos utilizadores várias opções que vão para lá da possibilidade de reproduzir músicas e que permitem customizar a experiência de cada utilizador. As várias funcionalidades vão desde a simples possibilidade de criar *playlists* individuais ou colaborativas até algoritmos de Aprendizagem Computacional para sugerir músicas aos utilizadores com base nos gostos.

Deste modo, foi necessário realizar uma filtragem das diversas funcionalidades que podiam ser incluídas no produto final, tendo em conta o tempo e os recursos disponíveis. As funcionalidades selecionadas para implementação foram modeladas utilizando um *use case diagram*, que se encontra na figura 1.

Finalmente, para fazer a GUI da aplicação do "pobrefy" recorreu-se novamente a uma biblioteca da linguagem de programação Python, o PYQT6. De entre as ferramentas utilizadas desta biblioteca é de referir o Qt Designer, uma ferramenta visual de design que facilita a criação da GUI por meio de uma abordagem intuitiva. A utilização do Qt Designer resulta na criação de arquivos descritivos da GUI (.ui), que podem ser posteriormente convertidos em código Python utilizando a ferramenta pyuic6. Assim, a combinação do PyQt6 com o Qt Designer ofereceu uma abordagem eficaz para o desenvolvimento da GUI da aplicação.

Como curiosidade, o nome "pobrefy" advém da ideia desta aplicação ser uma versão humilde das aplicações de *streaming* contemporâneas, em especial o Spotify.

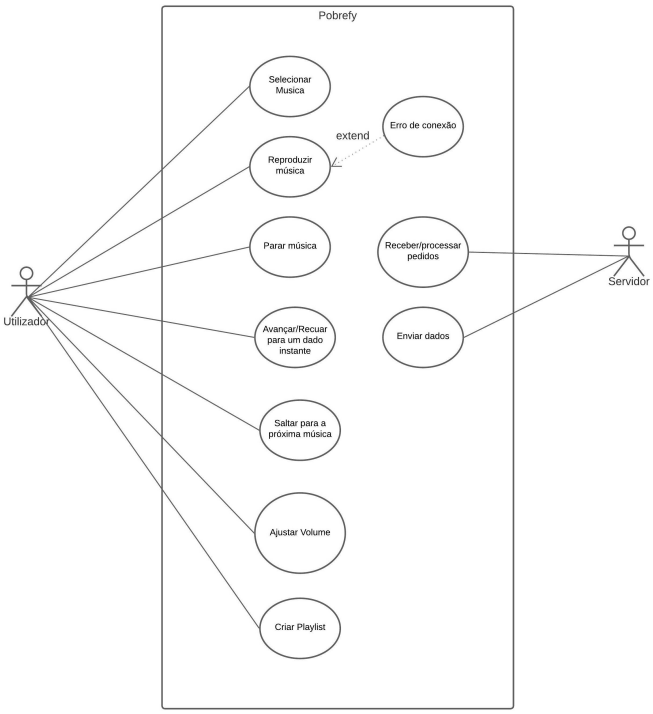

Figura 1. *Use case diagram* do "'pobrefy".

III. Implementação do Sistema

Na fase de implementação deste projeto, articularam-se todas as escolhas que tinham sido previamente realizadas de modo a desenvolver o sistema ilustrado na figura 2.

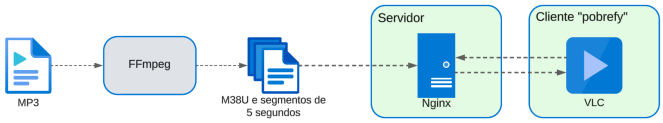

Figura 2. Arquitetura do "pobrefy".

A metodologia adotada para a implementação da aplicação inicia-se com uma fase de pré-codificação, na qual a música, originalmente codificada no formato MP3, é transcodificada utilizando a ferramenta FFmpeg, recorrendo ao comando apresentado na Figura 3. Durante este processo, a codificação é realizada no formato HLS, empregando o codec AAC com um *bitrate* de 128 Kbits/s. Apesar de não se obter a melhor qualidade de áudio, destaca-se pela capacidade de proporcionar uma experiência de serviço mais robusta em cenários nos quais a largura de banda é limitada, considerando, adicionalmente, que se trabalha com segmentos de 5 segundos. É importante realçar que, apesar do HLS suportar ABR, esta funcionalidade não foi implementada, sendo que o cliente só consegue receber áudio codificado a um *bitrate* de 128 Kbits/s, como tinha sido previamente mencionado.

O servidor Nginx encontra-se a correr numa máquina virtual e está disponível para receber pedidos e transmitir dados para

```
ruben@ruben-1-2:~$ ffmpeg -i input.mp3 -c:a aac -b:a 128k -hls_time 5 -hls_list_size 0 -hls_segment_
filename segment_%03d.ts Output.m3u8
```

Figura 3. Comando utilizado para transcodificar os ficheiros de música.

os clientes que se encontrem na rede local. O servidor Nginx estava inicialmente configurado para lidar com 768 clientes simultaneamente, contudo, devido ao facto do servidor estar a correr numa máquina virtual com um poder computacional reduzido, decidiu-se limitar o servidor a 10 clientes em simultâneo, dado que, nos testes realizados, a partir dos 15 clientes a reproduzir conteúdo diferente em simultâneo, o sistema começava a ficar lento.

Posto isto, os utilizadores podem iniciar a aplicação "pobrefy" através do ficheiro executável "pobrefy.exe" e, de seguida, irão deparar-se com uma lista de músicas de entre as quais podem selecionar uma para reprodução. Um pedido HTTP é enviado para o servidor Nginx de forma a adquirir o ficheiro M3U8 e, a partir daí, os segmentos da música vão sendo pedidos ao servidor conforme a necessidade de reprodução. Durante este processo, uma instância da classe vlc.MediaPlayer, denominada "player", recebe e descodifica os referidos ficheiros, preparando-os para subsequente reprodução. Além disso, de entre as funcionalidades de manipulação da reprodução pensadas inicialmente, conseguiu-se implementar a paragem e retoma da música, o avanço e recuo para um dado instante e o ajuste de volume. Este conjunto de interações entre o cliente e o servidor possibilita a experiência de *streaming* de áudio *on demand* na aplicação.

IV. Conclusão

Concluindo, este relatório detalha o desenvolvimento do projeto "pobrefy", um processador de áudio para *streaming on demand*. Desde a conceção à implementação, foram exploradas as escolhas efetuadas, nomeadamente a adoção do protocolo HLS, a transcodificação de MP3 para AAC, a seleção do servidor Nginx e a implementação de uma aplicação na linguagem de programação Python, recorrendo à biblioteca Python VLC, e sem esquecer o uso do PyQT6 para a GUI.

Apesar das limitações em relação aos serviços de *streaming* mais avançados, o "pobrefy" permite que o utilizador selecione e reproduza músicas *on demand* sem a necessidade de armazenamento local. Além disso, a aplicação apresenta algumas funcionalidades básicas de manipulação da reprodução.

Melhorias futuras incluem todos os *use cases* concebidos que não tiveram oportunidade de ser implementados, nomeadamente avançar ou retroceder de música e criar *playlists*.

Referências

[1] Multimedia streaming protocols Part 1: HAS concept and HLS (2023). FEUP.
[2] Multimedia streaming protocols Part 2: MPEG-DASH (2023). FEUP.
[3] Pantos, R. (2016). HTTP Live Streaming. https://datatracker.ietf.org/doc/html/draft-pantos-http-live-streaming-20
[4] Schulzrinne, H., Casner, S., Frederick, R., & Jacobson, V. (2003). RTP: A Transport Protocol for Real-Time Applications. https://datatracker.ietf.org/doc/html/rfc3550
[5] FFmpeg (2024). https://ffmpeg.org/
[6] Apache Software Foundation (2024). https://www.apache.org/

[7] NGINX (2024). https://www.nginx.com/
[8] python-vlc 3.0.20123 (2024). https://pypi.org/project/python-vlc/
[9] PyQt6 6.6.1 (2024). https://pypi.org/project/PyQt6/