Projeto 1 - Redes de Computadores - Streaming de Audio

Fabio Parra
Pedro Henrique
213214151
211020956

Universidade de Brasília

Abstract

Projeto desenvolvido com o objetivo de familiarizar-se com o funcionamento de sockets, que estabelecem a conexão entre a camada de aplicação e a camada de transporte para funcionamento de um aplicação de música, seguindo o modelo cliente-servidor. Além disso, explora-se os processos de comunicação entre um host e um servidor com objetivo.

Keywords: TCP, Socket, WireShark, Python

1. Introdução

A área de redes de computadores desempenha um papel essencial na conectividade e comunicação entre dispositivos e sistemas computacionais. Um aspecto crucial nesse contexto é o streaming de áudio, que possibilita a transmissão contínua de dados de áudio em tempo real pela rede.

Este trabalho tem como objetivo principal compreender o uso de sockets no contexto de redes de computadores, especificamente no paradigma cliente-servidor. Os sockets são interfaces de programação que estabelecem a comunicação entre a camada de aplicação e a camada de transporte em uma rede.

Ao utilizar o protocolo TCP (Transmission Control Protocol), que opera na camada de transporte do modelo OSI (Open Systems Interconnection), é possível implementar a comunicação confiável e orientada à conexão entre os processos de um host e um servidor.

A programação de sockets em Python oferece uma abordagem prática e eficiente para criar aplicações de rede, permitindo o desenvolvimento de sistemas de streaming de áudio robustos e escaláveis.

Neste relatório, apresentaremos a estrutura do trabalho, que inclui a compreensão dos conceitos fundamentais de sockets, o funcionamento da comunicação entre host e servidor, a interação da camada de transporte com a camada de aplicação e a programação de sockets em Python utilizando o protocolo TCP.

Contato dos autores:

email: 211020956@aluno.unb.br (Fabio Parra)
email: 202037604@aluno.unb.br (Pedro Henrique)

Para ampliar nosso conhecimento sobre redes de computadores e suas aplicações, buscaremos implementar sistemas de streaming de áudio. O presente relatório abordará uma análise minuciosa desses conceitos, além de apresentar exemplos práticos de implementação em Python, destacando a importância e as diversas possibilidades de aplicação do streaming de áudio no contexto das redes de computadores.

Com a estruturação do relatório delineada, abordaremos detalhadamente cada um dos aspectos mencionados, proporcionando uma visão abrangente e aprofundada sobre o tema proposto.

Nossa expectativa é que este trabalho contribua significativamente para a compreensão e aplicação dos conceitos relacionados ao uso de sockets para streaming de áudio, fornecendo uma base sólida para pesquisas futuras e aplicações práticas nesse campo promissor da computação e das redes de computadores.

2. Fundamentação Teórica

Nesta seção, abordaremos as ideias fundamentais relacionadas ao streaming de áudio, aos sockets e ao protocolo TCP, que são de extrema importância para a compreensão do tópico em questão. Vamos examinar cada um desses conceitos detalhadamente:

1. Streaming de áudio: O streaming de áudio consiste na transmissão contínua de dados de áudio em uma rede, permitindo que os usuários acessem e reproduzam conteúdo em tempo real, sem a necessidade de baixar um arquivo de áudio completo. Esse método de transmissão é amplamente utilizado em plataformas de música online, estações de rádio na Internet e outras aplicações de comunicação e entretenimento.

2. Sockets: Sockets são interfaces de programação que possibilitam a comunicação entre processos dentro de uma rede. Eles fornecem uma abstração de software para estabelecer conexões e trocar dados entre aplicativos em máquinas distintas.

3. Paradigma Cliente-Servidor: O modelo cliente-servidor é uma abordagem de comunicação em que um cliente solicita serviços de um servidor , que responde às solicitações enviando as informações solicitadas . No contexto do streaming de áudio , o servidor armazena e disponibiliza o conteúdo de áudio , enquanto o cliente requisita e recebe continuamente esse conteúdo para reprodução em tempo real.

4. Protocolo TCP: O Protocolo de Controle de Transmissão (TCP) é um protocolo de transporte confiável e orientado à conexão. Ele garante a entrega ordenada e livre de erros dos dados entre o cliente e o servidor . Além disso , o TCP controla a taxa de transmissão, evitando sobrecarregar a rede e assegurando que os dados sejam transmitidos e recebidos corretamente . O TCP é amplamente empregado em aplicações que requerem transmissão segura e confiável de dados , como o streaming de áudio , a transferência de arquivos e a navegação na web.

Programação de Sockets em Python: Python é uma linguagem de programação amplamente adotada para o desenvolvimento de aplicativos de rede. Através da biblioteca padrão do Python , podemos contar com suporte para programação de sockets, oferecendo classes e métodos que possibilitam a criação de conexões de rede , envio e recebimento de dados , e gerenciamento da comunicação entre cliente e servidor. Com a programação de sockets em Python , torna -se viável implementar aplicativos de streaming de áudio que se conectem e comuniquem eficientemente com servidores remotos.

Ao compreender e aplicar esses conceitos , estaremos preparados para explorar a implementação prática de sistemas de streaming de áudio utilizando sockets em Python com o protocolo TCP. Por meio desse conhecimento teórico e prático , seremos capazes de desenvolver soluções eficazes e escaláveis para transmitir áudio em tempo real através de redes de computadores.

A fundamentação teórica apresentada nesta seção fornecerá a base necessária para explorar os aspectos práticos e experimentais do streaming de áudio com sockets em Python, que serão abordados nas seções subsequentes deste trabalho. Com esse conhecimento sólido , estaremos preparados para aprofundar nosso entendimento e realizar experimentos que enriqueçam nossa compreensão sobre o funcionamento e aplicação do streaming de áudio em ambientes de redes de computadores.

3. Ambiente Experimental

3.1. Descrição do Cenário:

Para conduzir nosso experimento, optamos por utilizar um computador com sistema operacional Windows 10, desempenhando os papéis de servidor e cliente. A topologia de rede selecionada foi a topologia em estrela, na qual o servidor ocupa a posição central e o cliente é colocado como o nó periférico. Usamos os seguintes Software:

1. IDE Visual Studio Code (VSCode): utilizada como ambiente integrado de desenvolvimento para programar o projeto de streaming de áudio . O VSCode oferece recursos avançados de edição , depuração e gerenciamento de código.

2. Wireshark : Empregado para a captura de pacotes durante a transmissão de áudio . Essa ferramenta possibilitou a análise do tráfego de rede e a verificação da integridade dos pacotes transmitidos.

Além disso , no projeto , utilizamos as seguintes bibliotecas:

1. PyAudio: Utilizada para a reprodução do áudio no cliente , oferecendo uma interface para captura e reprodução de áudio em tempo real.

2. Socket : Implementada a biblioteca de sockets do Python para estabelecer a comunicação entre o servidor e o cliente. Os sockets permitem a troca de dados entre os dois pontos, viabilizando a transmissão contínua do áudio.

3. Time: Empregada para controlar o tempo de envio e recebimento das mensagens através dos sockets . O método sleep () foi utilizado para garantir que as mensagens enviadas por um socket não ocorressem simultaneamente.

4. Threading : Utilizada para criar threads separadas para o download da música e a reprodução simultânea . Essa abordagem permite que o cliente baixe e reproduza o áudio de forma assíncrona.

5. OS: Utilizada para localizar a música em uma pasta específica do servidor. Essa biblioteca oferece recursos para manipular diretórios, arquivos e caminhos.

A escolha do protocolo TCP para a transmissão de áudio foi motivada pela sua confiabilidade na entrega de dados . O TCP garante que os pacotes de áudio sejam recebidos na ordem correta e sem perdas , o que é essencial para uma experiência de streaming de áudio contínuo e sem interrupções.

A linguagem de programação escolhida foi o Python devido à sua facilidade de uso e à familiaridade do pesquisador com essa linguagem . Python oferece uma ampla variedade de bibliotecas e recursos que facilitam o desenvolvimento de aplicativos de streaming de áudio.

Essas escolhas e configurações proporcionaram um ambiente experimental coeso com os conceitos teóricos abordados e possibilitaram a implementação bem - sucedida do projeto de streaming de áudio.

3.2. Descrição da Configuração do Código do Servidor:

1. Inicialização do Servidor : O código do servidor inicia a execução criando um socket do tipo AF_INET e SOCK_STREAM e associa-o a um endereço IP e porta específicos . O servidor entra no modo de escuta para aceitar conexões de clientes.

2. Lidando com Clientes: Assim que um cliente se conecta ao servidor, o servidor imprime uma mensagem informando a conexão bem-sucedida com o endereço do cliente. A função lidar_cliente é iniciada em uma nova thread para tratar a interação com o cliente.

3. Opções de Serviço: A função lidar_cliente envia uma lista de opções de serviço disponíveis para o cliente e aguarda a resposta do cliente sobre qual serviço ele deseja utilizar.

4. Tocar Música: Se o cliente escolher a opção "1" para tocar música no seu dispositivo, ele recebe uma lista de músicas disponíveis no servidor e pode selecionar uma delas digitando o nome. O servidor verifica se a música escolhida está presente no cache local (na pasta "musicas") e envia a música em blocos de 30 segundos (ou conforme definido por CHUNK) para o cliente através da função donwload_musica. O cliente reproduz a música localmente e, ao final, fecha a conexão com o servidor.

5. Receber Música de Outros Clientes: Se o cliente escolher a opção "2" para receber músicas de outros clientes, ele é adicionado à lista de clientes ativos que estão disponíveis para enviar músicas. O servidor aguarda até que outro cliente escolha a opção de enviar música e encaminhe os pacotes da música para o cliente selecionado.

6. Escolher Cliente para Tocar Música: Se o cliente escolher a opção "3" para escolher outro cliente para tocar música, ele recebe uma lista de clientes conectados ao servidor e pode selecionar um cliente específico digitando o endereço IP e porta correspondentes. O servidor envia a lista de músicas disponíveis para o cliente selecionado e inicia a transmissão da música escolhida em blocos de tamanho determinado para o cliente escolhido.

7. Fechamento da Conexão: Após a conclusão das ações do cliente ou em caso de escolha de opções inválidas, a conexão com o cliente é fechada.

3.3. Descrição da Configuração do Código do Cliente:

1. Conexão com o Servidor: O código do cliente estabelece uma conexão com o servidor de streaming de áudio utilizando o endereço IP e a porta especificados. Essa conexão é essencial para a comunicação entre o cliente e o servidor durante toda a interação.

2. Recebendo Mensagens do Servidor: Após a conexão bem-sucedida, o cliente recebe uma mensagem de boas-vindas e uma lista de clientes online atualmente conectados ao servidor. Isso proporciona ao usuário uma visão geral dos clientes disponíveis e mostra que a conexão foi estabelecida com sucesso.

3. Opções de Serviço : O cliente recebe um conjunto de opções de serviço disponíveis e apresenta -as ao usuário . O usuário é solicitado a digitar o número correspondente à opção desejada. As opções incluem:
a) Reproduzir músicas do servidor.
b) Listar as músicas disponíveis no servidor.
c) Fazer upload de uma música para o servidor.

4. Reprodução de Músicas: Se o usuário escolher a opção de reproduzir músicas do servidor, o cliente apresenta a lista de músicas disponíveis e permite que o usuário selecione a música desejada digitando o nome dela. Em seguida, o cliente verifica se a música escolhida está armazenada no cache local utilizando a função CheckCache.

5. CheckCache: Se a música estiver no cache local, o cliente inicia a reprodução imediata da música usando a função PlayCache, evitando a necessidade de baixar a música novamente do servidor. Caso a música não esteja no cache local, o cliente solicita a música ao servidor usando a função PlayAndReceive, reproduzindo-a em tempo real e, simultaneamente, armazenando-a no cache local para uso futuro.

6. Encerramento e Desconexão: Ao concluir a reprodução da música escolhida, o cliente aguarda a finalização da reprodução usando a função join(), garantindo que a reprodução seja concluída antes de prosseguir.

Após a conclusão da reprodução ou em caso de escolha de outras opções de serviço , o cliente fecha a conexão com o servidor usando a função close() para liberar os recursos da rede.

4. Analises dos resultados:

4.1. Wireshark

Durante a observação das mensagens trocadas entre o servidor e o cliente, foram identificadas as seguintes informações, conforme demonstrado nas imagens da Figura 1:

A. Identificação da versão ou tipo de aplicação no servidor que está em execução é IPv4.

B. Caminho de dados criptografado por ser um protocolo tcp.

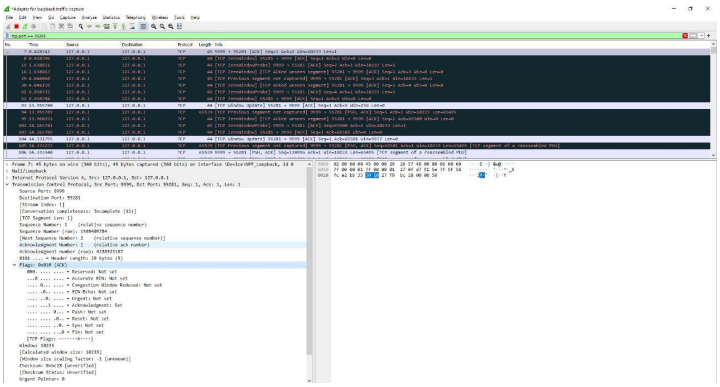

Figure 1 (A) : Imagens/telas impressas do Wireshark

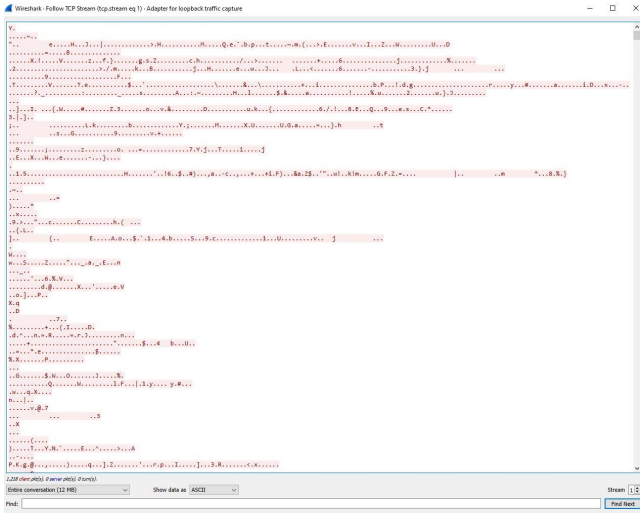

Figure 1 (B) : Imagens/telas impressas do Wireshark

5. Conclus˜ao

Após a realização deste trabalho , os autores puderam concluir que foi possível implementar com êxito um sistema de streaming de áudio utilizando uma topologia de rede em estrela.

Os aplicativos empregados , como o Visual Studio Code para programação , o Wireshark para captura de pacotes e análise de tráfego de rede, e as bibliotecas Python PyAudio , Socket , Time , Threading , Wave e OS , demonstraram -se adequados e eficazes no desenvolvimento do projeto de streaming de áudio.

A escolha de utilizar o protocolo TCP para a transmissão de áudio revelou -se prudente , visto que o TCP assegura a confiabilidade na entrega de dados , evitando perdas e garantindo uma transmissão contínua e ininterrupta do áudio.

Dessa forma , o projeto proporcionou um ambiente experimental coerente com os conceitos teóricos abordados , permitindo uma análise detalhada do funcionamento do streaming de áudio por meio de sockets em Python . A implementação bem -sucedida desse sistema reforça a importância e a viabilidade do uso de sockets para comunicação entre dispositivos e aplicações em redes de computadores . Com base nessa experiência , abre-se um leque de possibilidades para futuros estudos e aplicações no campo do streaming de áudio e das redes de computadores.

Bibliografia

1. Biblioteca OS. disponível em:
https://docs.python.org/pt-br/3/library/os.html
2. Biblioteca Pyaudio. disponível em:
https://pypi.org/project/PyAudio/
3. Biblioteca Socket disponível em:
https://docs.python.org/3/library/socket.html
4. Biblioteca Thread disponível em:
https://docs.python.org/3/library/socket.html
5. Biblioteca time disponível em:
https://docs.python.org/3/library/time.html
6. Python disponível em:
https://www.python.org/doc/
7. Repositório:
https://www.python.org/doc/