Projeto 1 - Redes de Computadores - Streaming de Áudio

Luiz Fernando Sperandio David
190112735
Turma 01

[a] *Universidade de Brasília, Brasília, Brazil*

Abstract

Projeto realizado com proposito de compreender o uso de sockets, que conecta a camada de aplicação a camada de transporte, em cima do paradigma cliente-servidor. Assim como os processos de comunicação entre host e servidor. Como a camada de transporte interage com a camada de aplicação. Como programar sockets em python com o protocolo TCP.

Keywords: TCP, Socket, WireShark, Python, Streaming Audio TCP

1. Introdução

A área de redes de computadores desempenha um papel fundamental na conectividade e comunicação entre dispositivos e sistemas computacionais. Um dos aspectos importantes dentro desse contexto é o streaming de áudio, que permite a transmissão contínua de dados de áudio em tempo real pela rede.

Este trabalho tem como objetivo compreender o uso de sockets no contexto de redes de computadores, mais 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 proporciona uma maneira prática e eficiente de criar aplicações de rede, permitindo o desenvolvimento de sistemas de streaming de áudio robustos e escaláveis.

Neste relatório, apresentaremos a estruturação do trabalho, que abrange 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. Ao explorar esses conceitos e realizar experimentos práticos, buscaremos aprofundar nosso conhecimento sobre o uso de sockets para implementar sistemas de streaming de áudio, ampliando assim nosso entendimento sobre as redes de computadores e suas aplicações.

Dessa forma, este relatório apresentará uma análise detalhada desses conceitos, além de fornecer exemplos práticos de implementação em Python, evidenciando a relevância e as possibilidades de aplicação do streaming de áudio no contexto das redes de computadores.

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

Esperamos que este trabalho contribua para o entendimento e a aplicação dos conceitos relacionados ao uso de sockets para streaming de áudio, fornecendo uma base sólida para futuros estudos e aplicações nesse campo promissor da computação e das redes de computadores.

2. Fundamentação Teórica

Nesta seção, discutiremos as ideias fundamentais relacionadas ao streaming de áudio, aos sockets e ao protocolo TCP que são essenciais para a compreensão do tópico em questão. Examinaremos cada um deles em detalhes.

1. Streaming de áudio: Streaming de áudio é a transmissão contínua de dados de áudio em uma rede que permite aos usuários acessar e reproduzir conteúdo em tempo real sem a necessidade de concluir o download de um arquivo de áudio completo. Este 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 permitem a comunicação entre processos dentro de uma rede. Eles fornecem uma abstração de software

*Corresponding author
Email address: 190112735@aluno.unb.br (Luiz Fernando Sperandio David
190112735
Turma 01)

para comunicação de rede que permite que aplicativos em vários dispositivos se comuniquem entre si por meio de uma conexão de rede. No aplicativo OSI e nas vias de transporte, os sockets são implementados.

3. Paradigma Cliente-Servidor: O paradigma cliente-servidor é um modelo de comunicação em que um cliente solicita um serviço a um servidor, que por sua vez fornece a resposta apropriada. No contexto de streaming de áudio, o servidor armazena o conteúdo de áudio e o cliente solicita a transmissão contínua desse conteúdo para reprodução em tempo real.
4. Protocolo TCP:
   O Transmission Control Protocol (TCP) é um protocolo de transporte confiável e orientado à conexão. Ele garante que os dados sejam entregues de forma sequencial, sem perdas ou corrupção, e também controla o fluxo de dados entre o cliente e o servidor. O TCP é amplamente utilizado em aplicações que exigem uma entrega confiável, como streaming de áudio, transferência de arquivos e navegação na web.

Programação de Sockets em Python: Python é uma linguagem de programação amplamente utilizada para o desenvolvimento de aplicativos de rede. A biblioteca padrão do Python oferece suporte à programação de sockets, fornecendo classes e métodos para criar conexões de rede, enviar e receber dados e gerenciar a comunicação entre o cliente e o servidor. Através da programação de sockets em Python, é possível implementar aplicativos de streaming de áudio que se conectam e se comunicam eficientemente com servidores remotos.

Ao compreender e aplicar esses conceitos, estaremos preparados para explorar a implementação prática de sistemas de streaming de áudio usando sockets em Python com o protocolo TCP. Através desse conhecimento teórico e prático, poderemos criar soluções eficazes e escaláveis para transmitir áudio em tempo real por meio 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 discutidos nas seções subsequentes deste trabalho.

3. Ambiente Experimental

3.1. Descrição do Cenário:

Para o nosso experimento, utilizamos um computador com sistema operacional Windows 10 como servidor e cliente. A topologia de rede adotada foi em estrela, em que o servidor é o ponto central e o cliente é o nó periférico. A escolha dessa topologia permite uma conexão direta entre o servidor e o cliente, facilitando a transmissão dos dados de áudio. Imagens na Figura 1

Os softwares utilizados foram os seguintes:

1. VSCODE: Utilizamos o Visual Studio Code como ambiente de desenvolvimento integrado (IDE) para programar o projeto de streaming de áudio. Ele oferece recursos de edição, depuração e gerenciamento de código.
2. Wireshark: Utilizamos o Wireshark para realizar a captura de pacotes durante a transmissão de áudio. Essa ferramenta nos permitiu analisar o tráfego de rede e verificar a integridade dos pacotes transmitidos.

Além disso, as seguintes bibliotecas foram utilizadas em nosso projeto:

1. PyAudio: Utilizamos a biblioteca PyAudio para reproduzir o áudio no cliente. Ela fornece uma interface para captura e reprodução de áudio em tempo real.
2. Socket: Utilizamos 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, facilitando a transmissão contínua do áudio.
3. Time: Utilizamos a biblioteca time para controlar o tempo de envio e recebimento das mensagens por meio dos sockets. O método sleep() foi usado para garantir que uma mensagem enviada por um socket não chegasse simultaneamente a outra mensagem.
4. Threading: Utilizamos a biblioteca threading 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. Wave: Utilizamos a biblioteca wave para analisar os dados do arquivo de áudio. Essa biblioteca nos permitiu obter informações sobre o formato, duração e outros atributos do arquivo de áudio.
6. OS: Utilizamos a biblioteca os para buscar a música em uma pasta específica do servidor. Essa biblioteca oferece recursos para manipular diretórios, arquivos e caminhos.

A escolha da camada TCP na transmissão de áudio se deu 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 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 coerente com os conceitos teóricos abordados e permitiram a implementação bem-sucedida

do streaming de áudio em nosso projeto.

Descrição da Configuração do Servidor:
Imagens na Figura 2

- A configuração do servidor inicia com a definição do endereço IP do servidor na variável ("endereco servidor") a porta na variável ("porta servidor").
- A variável ("max conexoes") define o número máximo de conexões simultâneas permitidas pelo servidor.
- O servidor cria um socket utilizando a função "socket .socket()" especificando o tipo de endereço ("AF INET") e o tipo de socket ("SOCK STREAM").
- O socket do servidor é vinculado ao endereço e porta especificados usando a função bind().
- O servidor entra em um estado de escuta passiva através da função listen(), que aguarda conexões de clientes.
- Em seguida, é exibida uma mensagem indicando que o servidor foi iniciado e está aguardando novas conexões.
- O dicionário "dict dispositivos sockets" é inicializado para mapear endereços IP de dispositivos conectados aos seus respectivos sockets.
- A lista "dispositivos conectados" é inicializada para armazenar os endereços IP e portas dos dispositivos conectados.
- O servidor entra em um loop infinito usando o "while True" para aguardar e aceitar novas conexões de clientes.
- Quando uma conexão é estabelecida com sucesso, o método accept() é chamado no socket do servidor, retornando um objeto socket do cliente e o endereço do cliente.
- O endereço IP do cliente é usado como chave no dicionário "dict dispositivos sockets" para mapear o socket correspondente.
- O endereço IP e a porta do cliente são adicionados à lista "dispositivo conectados" para fins de registro.
- Uma nova thread é criada para lidar com as operações do cliente, com o método clienttread() sendo chamado e passando o socket do cliente e o endereço do cliente como argumentos.
- A thread é iniciada chamando o método start().

A seguir uma exemplificação do funcionamento dos serviços:

A parte principal do código do servidor é a função clienttread. Essa função é executada em um thread separado para lidar com cada cliente que se conecta ao servidor. Aqui está uma descrição de como esta parte do código funciona:

A função clienttread usa o socket do cliente e o endereço do cliente como parâmetros. Ele tem um loop que é executado até que seja definido como "concluído". Dentro do loop, a função recebe os dados enviados pelo cliente através do método recv do socket. Os dados recebidos são tratados de acordo com a sua natureza.

- A função sendListaMusicas é utilizada para enviar ao cliente uma lista das músicas que já estão disponíveis se os dados forem iguais a "lista" ou "7".
- A função checkExisteMusica é usada para determinar se a música especificada está presente no servidor se os dados começarem com "download". Se a música estiver presente, a função chamada "download Music Client" é invocada para enviar os dados da música para o cliente.
- Se os resultados forem iguais a "9" ou "quit", o loop é fechado e a conexão com o cliente é encerrada. Além disso, o dispositivo do cliente é eliminado da lista de dispositivos conectados.
- A função sendListaDispositivos é utilizada para enviar ao cliente uma lista de dispositivos conectados se os dados forem iguais a "lista dispositivos" ou "10".
- O status do dispositivo do cliente é atualizado na lista de dispositivos conectados se os dados começarem com "att status".
- Caso nenhum dos casos listados acima coincida com os dados fornecidos, uma mensagem informando "Comando Inexistente" é enviada de volta ao cliente.

Descrição da Configuração do Cliente:

- A configuração do cliente inicia com a criação de um socket utilizando a função socket.socket() e especificando o tipo de endereço ("AF INET") e o tipo de socket (SOCK STREAM).
- O socket do cliente é conectado ao endereço IP do servidor e à porta especificados usando o método connect(). O nome do cliente é obtido utilizando a função gethostname() e o IP do cliente é obtido usando a função gethostbyname().
- Em seguida, é exibido um menu interativo para o usuário com uma série de opções de comandos para controlar a reprodução de áudio.
- O cliente aguarda a entrada do comando do usuário e realiza a ação correspondente com base no comando fornecido.
- As ações incluem pausar, retomar, reiniciar, parar, repetir, reproduzir, listar músicas, alterar música, listar dispositivos e sair.
- O cliente envia os comandos e recebe as respostas do servidor utilizando os métodos sendDados() e receberDados().
- O loop continua até que o usuário selecione a opção de sair, momento em que a conexão com o servidor é encerrada e o programa é encerrado.

A seguir uma exemplificação do funcionamento dos serviços:

A parte principal do código é o "while not feito" , onde ocorre a interação do usuário. Um menu é exibido e o usuário pode inserir comandos para gerenciar a reprodução da música. Play, resume, restart, stop, loop, pause, lista, change, lista dispositivos e quit são alguns dos comandos disponíveis. Dependendo do comando inserido, a ação apropriada é executada.

- A função chamada tocarMusica que é executada em uma thread separada. Ela recebe o nome da música como parâmetro e reproduz a música a partir dos dados armazenados em cacheLocal. A função utiliza a biblioteca pyaudio para reproduzir os frames da música.
- A função carregarMSC encarrega-se de pedir ao servidor para obter música através do socket e manda a mensagem "track data start" para o server onde analisaremos no wireshark para conseguirmos ver onde começa a musica. A função baixarMusica é então executada, recebendo a música em blocos de 30 segundos e armazenando-a no cacheLocal até que toda a música seja recebida e recebe a mensagem "track data end" onde analisaremos também no wireshark para identificar onde o bloco da musica em data acaba.
- O código possui algumas funções auxiliares como sendDados e receiveDados que se encarregam de enviar e receber mensagens via socket.A função getListaMsc, que solicita a lista de músicas ao servidor e armazena em listaCacheLocal, e outras funções e variáveis utilizadas no programa.

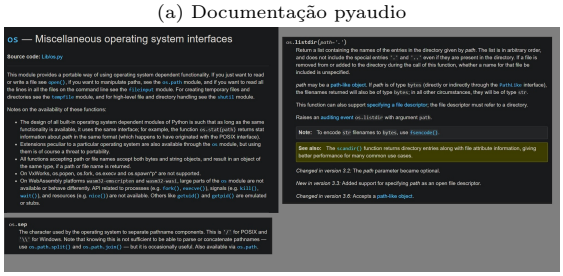
4. Análise de Resultados

4.1. Wiresharp

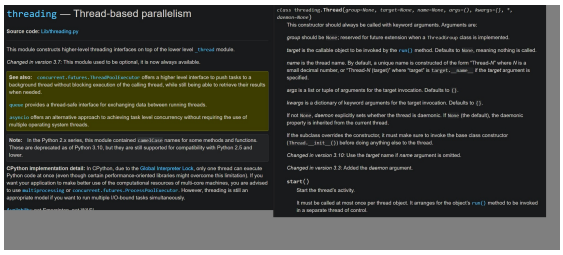
Ao observar as informações nas mensagens entre o servidor e o cliente, podemos tirar as informações das imagens na Figura 3 e iremos listar aqui as mesmas:

A. Identificação da versão ou tipo da aplicação no servidor que está executando é IPv4. (Figura 3a)
B. O endereço IP do servidor é 10.0.0.7 e do cliente é 10.0.0.10. (Figura 3a)
C. O protocolo de transporte usado é TCP. (Figura 3a)
D. A porta de destino do cliente é 2635 e a porta de origem do cliente é 45926. (Figura 3a)
E. A carga útil identificada na Figura 3b é "listaRVVLquitgoodbye", que corresponde ao esperado para a aplicação desenvolvida.

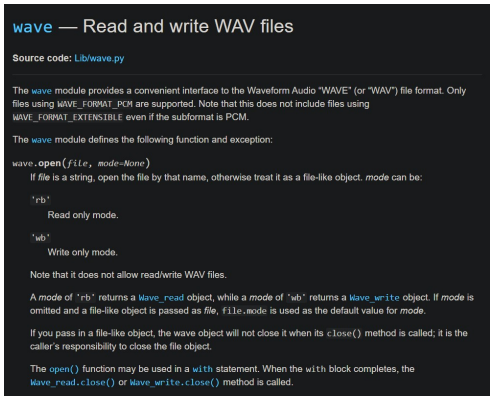
(a) Documentação pyaudio



(b) Documentação OS



(c) Documentação threading



(d) Documentação wave

Figure 1: Documentações das Bibliotecas

5. Conclusão

Com base no trabalho realizado, os autores concluíram que era possível uma implementação bem-sucedida de um sistema de streaming de áudio usando uma topologia de rede em estrela.

Os aplicativos utilizados, como Visual Studio Code para programação, Wireshark para captura de pacotes e análise de tráfego de rede, e as bibliotecas Python PyAudio, Socket, Time, Threading, Wave e OS, mostraram-se adequados para o desenvolvimento do streaming de áudio projeto.

A decisão de usar o TCP para transmissão de áudio foi considerada sábia porque o TCP garante a confiabilidade da entrega de dados, evita a perda de dados e garante a transmissão ininterrupta.

```
while not feito:
    print("_____\n|
    comando = input("Digite um comando: \n").lower()
    '''
    print("_____")
    print("|   1.Play        2. Resume     3. Restart    4. Stop     |")
    print("|        5. Loop       6. Pause      7. Lista             |")
    print("|         8. Change(msc)       9. List_Dipo              |")"
    print("|_____10. Quit_____|")
    '''
    if comando == "pause" or comando == "6":
        if play:
```

(a) Interface com Usuario

```
ip = obter_ip()

endereco_servidor = ip
porta_servidor = 2635
max_conexoes = 5
socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
socket_servidor.bind((endereco_servidor, porta_servidor))
socket_servidor.listen(max_conexoes)
print("Iniciando servidor. Aguardando novas conexões\n ... ")
print(ip)
dict_dispositivos_sockets = {}
dispositivos_conectados = []

while True:
    try:
        (socketCliente, enderecoCliente) = socket_servidor.accept()
        dict_dispositivos_sockets[enderecoCliente[0]] = socketCliente
        dispositivos_conectados.append(
            [enderecoCliente[0], enderecoCliente[1]])
    except socket.timeout:
        print(f"Servidor: Desligando thread de escuta")
        break
    tserver = Thread(target=clienttread, args=(
        socketCliente, enderecoCliente), daemon=True)
    tserver.start()
```

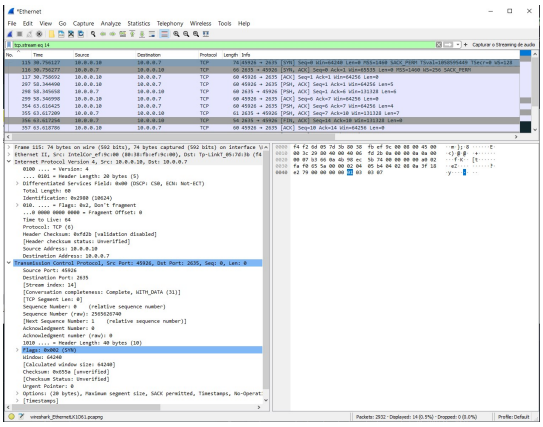
(b) Descrição dos elementos básicos da configuração do cliente e servidor

Figure 2: Interface e Elementos da Configuração

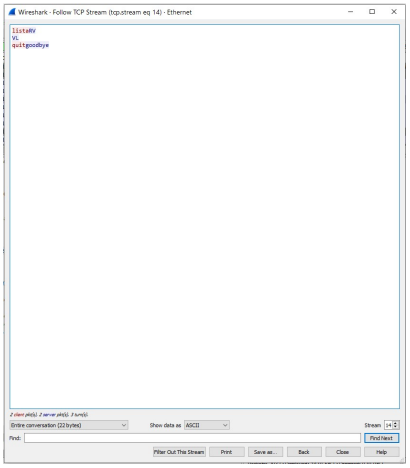
A utilização da linguagem de programação Python se mostrou vantajosa devido à sua facilidade de uso e à disponibilidade de bibliotecas e recursos que facilitaram o desenvolvimento do aplicativo de streaming de áudio.

Por fim, o autor destaca que as configurações realizadas, tanto no servidor quanto no cliente, foram adequadas para o funcionamento correto do sistema, permitindo a conexão entre os dispositivos, a transmissão e reprodução de áudio de forma assíncrona, e o controle das operações por meio dos comandos disponíveis no menu interativo.

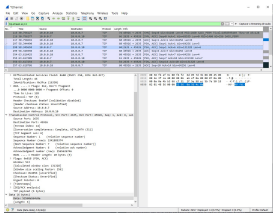
Em resumo, o trabalho demonstrou a viabilidade da implementação de um sistema de streaming de áudio com sucesso, com base nas escolhas de topologia de rede, softwares utilizados e configurações realizadas. Os resultados obtidos confirmam a eficácia do sistema proposto.



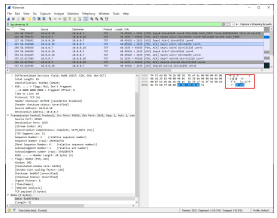
(a) Wireshark Dados



(b) carga útil dos pacotes



(c) Wireshark dados adicionais



(d) Wireshark dados adicionais

Figure 3: Imagens/telas impressas do Wireshark

Bibliografia

1. Biblioteca wave. Disponível em: `https://docs.python.org/3/library/wave.html`
2. Biblioteca pyaudio. Disponível em: `https://people.csail.mit.edu/hubert/pyaudio/docs/`
3. Biblioteca OS. Disponível em: `https://docs.python.org/3/library/os.html`
4. Biblioteca Thread. Disponível em: `https://docs.python.org/3/library/threading.html`
5. Biblioteca python. Disponível em: `https://docs.python.org/pt-br/3.9/library/time.html`
6. How to send audio data using socket programming in Python. Disponível em: `https://pyshine.com/How-to-send-audio-from-PyAudio-over-socket/`
7. Biblioteca time em: `https://docs.python.org/pt-br/3.9/library/time.html`

Visite o repositório do projeto de streaming de áudio no GitHub:
https://github.com/Sedinha/Redes-Projeto-Streaming-de-Audio

Vídeo de demonstração:
https://youtu.be/c3Thz4lUzjE