Dropbox interno

Projeto de aplicação de transferência de arquivos desenvolvido na disciplina Redes de Computadores

Rafael Dias Ghiorzi
232006144

Luiz Augusto Araujo da Silva
211036105

***Abstract*—This document is the project report for the Networks course at the University of Brasília. An application for file transfer between users was developed using knowledge of Socket programming, the client-server paradigm, network systems, and HTTP requests.**

***Abstract*—Esse documento é o relatório do projeto da disciplina de Redes da Universidade de Brasília. Foi desenvolvido uma aplicação de transferência de arquivos entre usuários utilizando o conhecimento sobre programação com Sockets, Paradigma cliente e servidor e sistema de redes e requisições HTTP.**

***Index Terms*—Socket, client-server, HTTP, protocol**

I. Introdução

O projeto consiste na construção de um sistema de transferência de arquivos entre usuários de uma rede local, usando os conceitos ministrados na disciplina Redes de Computadores. Foi necessário o desenvolvimento de um protocolo de transmissão de dados, assim como o uso de programação com sockets para estabelecer uma conexão entre Clientes e Servidor. O programa permite o cadastro e login de usuários, que por sua vez podem fazer o upload e download de qualquer tipo de arquivo presente em um servidor único.

II. Fundamentação teórica

Durante o desenvolvimento do projeto, os conceitos desenvolvidos com base, principalmente, nas camadas de aplicação e transporte da internet foram usados como fundamentação teórica. Ademais, foi colocado em prática uma aplicação que segue o paradigma cliente-servidor, fazendo o uso de Sockets na linguagem Python, que nos permite fazer a comunicação entre diferentes clientes a um único servidor, desde que seja passado o endereço IPv4 e a porta em que o servidor está ativo e ambos cliente e servidor estejam conectados à mesma rede. Os sockets usados seguiam o Transfer Control Protocol (TCP) para transporte, dado que esse protocolo nos garante a transmissão de pacotes de uma fonte a um destino sem perdas significativas. A troca de informações entre o cliente e o servidor se dava por uma aplicação simplificada do protocolo HTTP, em que a mensagem era separada por corpo e cabeçalho. O cabeçalho continha informações relacionadas aos métodos padrão GET, POST e DELETE e a função desejada, representada por uma rota, assim como o tamanho da mensagem enviada, caso se tratasse de upload de arquivos. O corpo contém as informações necessárias para o funcionamento das funções. O servidor, enquanto estiver ativo, mantém um socket aberto esperando a conexão com clientes, que por sua vez, mantinham ativo uma conexão com soccket apenas durante o processo de transmissão de dados, com o objetivo de não sobrecarregar o servidor com muitas conexões abertas ao mesmo tempo.

III. Ambiente experimental e análise de resultados

A. Descrição do cenário

A aplicação foi desenvolvido utilizando apenas a linguagem Python e algums bibliotcas e pacotes desse linguagem. Para a comunicação entre cliente e servidor, foi usado *sockets*, a partir da biblioteca Socket, configurados para seguir com o padrão TCP. Para comunicação com o banco de dados, foi utilizada a biblioteca SQLite3. Para que

1) Configuração do Cliente: Antes do cliente ser iniciado, é inicializado um socket com o **AF_INET**, para requisitar conexões utilizando o IPv4, e com **SOCK_STREAM** para usar **TCP** como o protocolo de transporte desse socket. É feito então uma requisição sem informações, apenas para checar se o servidor está offline. Caso a requisição receba uma resposta, o programa é iniciado e o socket é fechado. A partir daí, esse mesmo socket é aberto apenas quando é necessário fazer uma requisição e enviar uma mensagem para o servidor. Todas as requisições recebem de volta uma mensagem contendo alguma informação útil como, por exemplo, o *username* que é armazenado em uma variável global no programa quando o usuário faz o login para acessar os serviços do sistema. No caso de uma requisição de download, o cliente recebe as informações de um certo arquivo em pacotes e depois o armazena em uma pasta chamada downloads, dentro do diretório do arquivo python do cliente. O sistema foi desenvolvido pensando no paradigma de programação procedimental, em que o código é estruturado em procedimentos que levam em conta a vontade do usuário.

2) Configuração do Servidor: Uma vez que o servidor está rodando a aplicação, ele espera em um loop por uma nova conexão em seu socket (definido da mesma forma que o socket do cliente). Quando é recebido uma conexão, o servidor interpreta a mensagem contida na requisição e realiza as funções pedidas. No caso de um upload de arquivos, acontece da mesma forma que o download do cliente, e o arquivo final é armazenado em uma pasta chamada uploads no mesmo diretório dos arquivos python do servidor. Essas funções podem ser tanto de criação ou remoção de dados quanto para a transferência de informações. Esse loop cria uma

thread para cada cliente, evitando problemas quando várias requisições são realizadas ao mesmo tempo. As informações dos usuários e os metadados dos arquivos guardados no servidos são armazenados em um banco de dados SQLite, que é administrado pelo sistema utilizando a biblioteca *SQLite3*.

3) Formatação de mensagens: Como mencionado, a aplicação faz o uso de uma espécie de HTTP simplificado. O protocolo HTTP faz uso de cabeçalhos que explicitam a intenção e tipo de cada requisição e um corpo que contém informações necessárias para as requisições que necessitam. Dessa forma, a aplicação se baseia nessas regras, e cada requisição contém necessariamente um cabeçalho com o tipo de requisição (GET,POST,DELETE) assim como uma rota que direciona à função que ela quer executar (/user/register, por exemplo). O cabeçalho é separado por uma quebra de linha dupla e o corpo é enviado em formato JSON, podendo conter ou não informações. Para as funções de download e upload de arquivos, as requisições também contém o tamanho em bytes do arquivo e *boundaries* que delimitam diferentes partes da requisição. As respostas do servidor também seguem o padrão HTTP, contendo sempre um código de status, uma mensagem e um corpo separado por quebra de linha dupla.

4) Funcionalidades do usuário: O usuário do programa contém apenas um *username*, senha e uma lista de arquivos que ele fez o upload para o servidor. As funcionalidades são ass seguintes: registro, login, listagem e remoção de usários e upload, download, remoção e listagem de arquivos. A checagem de existência de duplos usuários ou arquivos é feita durante o processo de inserção no banco de dados, dado que a ferramenta SQLite já possui essa funcionalidade naturalmente, assim não sendo necessário a implementação de checagem de elementos repetidos. O usuário pode acessar o sistema ao inserir seu *username* e senha na página de login caso ele já tenha uma conta, ou diretamente pela página de registro, usando os mesmos dados, caso ainda não exista uma conta com essas informações.

5) Interface Gráfica: Tendo em mente a facilidade de uso da aplicação, o sistema foi todo desenvolvido para funcionar no terminal de comando do computador. A navegação do cliente é inteiramente feita a partir de entradas no teclado, enquanto o servidor roda sem interferência de ninguém. Uma vantagem do terminal de comando é sua universalidade, podendo rodar em qualquer computador com mínima ou nenhuma diferença entre diferentes sistemas operacionais. As imagens da interface do cliente estão a seguir:

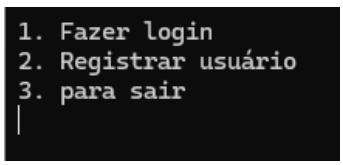

Fig. 1. Menu inicial do programa

6) Funcionamento da aplicação: Com o objetivo de demonstrar o funcionamento da aplicação, foi feito um **vídeo no youtube**, o qual exemplifica as funcionalidades e os

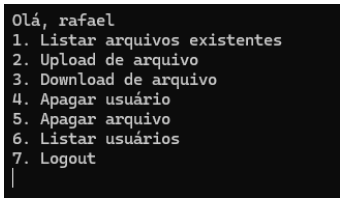

Fig. 2. Menu do usuário logado, seguido de suas funcionalidades

resultados obtidos para o experimento proposto pelo projeto. Além disso, está disponível o link para o repositório com o código-fonte do projeto no GitHub.

B. Análise de Resultados

No roteiro do relatório, foram passados 6 perguntas, as quais serão respondidas a seguir:

*1) **A**:* Aplicação Dropbox Interno, versão 1.0.1.

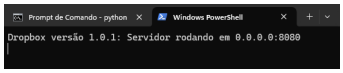

Fig. 3. Versão do servidor da aplicação

*2) **B**:* Analisando uma requisição, o IP do cliente (source) era 192.168.1.3 e o IP do servidor (Destination) era 192.168.106.

Source	Destination
192.168.1.3	192.168.1.106
192.168.1.3	192.168.1.106
192.168.1.3	192.168.1.106
192.168.1.106	192.168.1.3
192.168.1.3	192.168.1.106
192.168.1.106	192.168.1.3
192.168.1.3	192.168.1.106
192.168.1.3	192.168.1.106

Fig. 4. Endereço IP do servidor e cliente

*3) **C**:* Analisando os mesmos pacotes da requisição da imagem 4, é possível observar que o protocolo de transporte utilizado é o **TCP**.

Protocol
TCP
TCP
TCP
TCP

Fig. 5. Protocolo dos pacotes enviados

*4) **D**:* Ao analisar os pacotes, pode-se observar que a porta do servidor é sempre 8080, como definido no código. Já o cliente faz uso de portas dinâmicas. Dado que elas são abertas em momentos aleatórios, o sistema operacional procura por

portas disponíveis acima da porta 5000. No caso do pacote analisado, o cliente está na porta 60576.

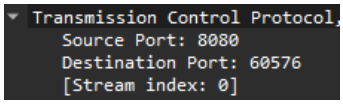

Fig. 6. Descrição das portas de origem e destino

*5) **E**:* O tamanho dos pacotes enviados se limitam a 65539 bytes. O socket estava programado para receber 4096 bytes, e ao analisar o monitoramento da ferramenta WireShark, percebe-se que, apesar de não terem exatamente 4096 bytes (o wireshark junta alguns pacotes, por isso eles parecem maiores), todos os pacotes tem exatamente o mesmo tamanho, 65539 bytes (que pode ser observado no trecho ”len=65539”), o que demonstra que a padronização de envio de dados está funcionando corretamente.

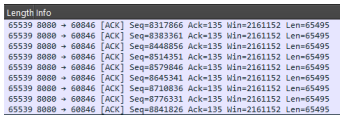

Fig. 7. Informações gerais dos pacotes enviados

*6) **F**:* Analisando os dados de um pacote escolhido arbitrariamente, podemos analisar os seguintes cabeçalhos e dados (indicados em vermelho):

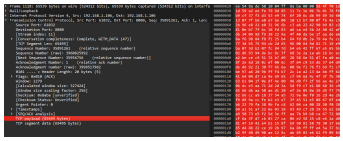

Fig. 8. Dados brutos do conteúdo da mensagem

Fig. 9. Dados brutos do cabeçalho da camada de transporte

Observando os dados brutos, é fácil reconhecer a pilha de protocolos da internet. Cada camada adiciona um cabeçalho com dados essenciais para o transporte de dados de uma máquina até outra. Cada camada recebe e trata seus respectivos cabeçalhos para que a mensagem seja recebida com segurança. Curiosamente, percebe-se que o cabeçalho da camada de enlace nessa aplicação é nulo. Isso acontece porque a aplicação está funcionando localmente, em uma mesma rede, portanto, as mensagens não são transmitidas para outro lugar na rede, e não passam pela camada de enlace.

IV. Conclusões

Concluímos este trabalho implementanddo todos os requisitos apresentados no roteiro para um aplicativo de transfferência de arquivos. Durante o processo, a implementação de sockets

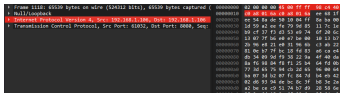

Fig. 10. Dados brutos do cabeçalho da camada de rede

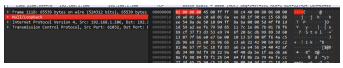

Fig. 11. Dados brutos do cabeçalho da camada de enlace

seguindo o protocolo TCP e de uma aplicação cliente-servidor trouxe a oportunidade de aplicar os conceitos lecionados ao decorrer do semestre. Foi possível aprender bastante sobre as tecnologias aplicadas em milhares de sistemas todos os dias.

References

[1] Mozilla, “HTTP Overview,” https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview, acessado em 18 de janeiro de 2025.
[2] Wireshark, “Learn Wireshark,” https://www.wireshark.org/learn, acessado em 18 de janeiro de 2025.
[3] I. Nando, “Group Chat Project,” repositório GitHub, https://github.com/isasisnando/Group_Chat_Project/tree/main, acessado em 18 de janeiro de 2025.
[4] R. Ghiorzi, “Dropbox-Redes-UnB,” repositório GitHub, https://github.com/rafaelghiorzi/Dropbox-Redes-UnB, acessado em 18 de janeiro de 2025.
[5] Python Software Foundation, “socket — Low-level networking interface,” https://docs.python.org/3/library/socket.html, acessado em 18 de janeiro de 2025.