UNIVERSIDADE FEDERAL DE ALAGOAS

INSTITUTO DE COMPUTAÇÃO/IC

REDES DE COMPUTADORES

RELATÓRIO -TRANSFERÊNCIA DE ARQUIVOS VIA SOCKET

DISCENTES:

Alan Diogo da Rocha Oliveira

Gabriel Lisboa Conde da Rocha

João Vitor Alves da Silva

Lucas de Amorim Caldas Alves da Silva

NOVEMBRO DE 2025

MACEIÓ - AL

1. INTRODUÇÃO

Este relatório detalha o sistema de Transferência de Arquivos via Socket desenvolvido para a disciplina de Redes de Computadores. O projeto, implementado em Python, foca na concorrência: o servidor utiliza threading (um requisito obrigatório) para gerenciar vários uploads de clientes simultaneamente. A aplicação permite que clientes enviem arquivos para um diretório dedicado (Arquivos Recebidos) usando comunicação TCP.

2. ARQUITETURA E PROTOCOLO

Característica	**Detalhe da Implementação**	**Justificativa**
Protocolo de Transporte	TCP (Socket Stream - `SOCK_STREAM`)	Garante a entrega confiável e ordenada dos dados, essencial para a transferência de arquivos.
Protocolo de Rede	IPv4 (`AF_INET`)	Padrão para endereçamento na rede local e internet.
Concorrência	Múltiplas Threads	O servidor utiliza `threading.Thread` para lidar com cada conexão de cliente em paralelo, permitindo que o servidor aceite novas conexões enquanto processa transferências ativas.
Comunicação	Sockets Primitivos	Utilização direta das funções `socket()` , `bind()` , `listen()` , `accept()` , `connect()` , `sendall()` , e `recv()` , conforme exigido.

3. FLUXO DE TRANSFERÊNCIA DE ARQUIVOS

O protocolo de transferência sobre TCP segue os seguintes passos:

1. **Handshake:** O cliente envia o nome do arquivo. O servidor verifica a existência e responde com OK (permitido) ou EXISTS (arquivo já existe).
2. **Metadados:** Se OK, o cliente envia os metadados no formato nome:tamanho.
3. **Transferência:** O cliente envia os dados do arquivo em blocos (BUFFER_SIZE).

4. **Confirmação:** O servidor recebe os blocos, salva o arquivo e verifica se o tamanho recebido bate com os metadados.

4. O QUE PODERIA TER SIDO IMPLEMENTADO A MAIS

Embora o projeto atenda aos requisitos mínimos, identificamos diversas melhorias que poderiam ser implementadas para tornar a aplicação mais robusta e completa:

Interface de Usuário (GUI ou CLI Aprimorada): Atualmente, a interação é via linha de comando. Uma interface gráfica ou uma CLI mais interativa (com status de progresso, por exemplo) melhoraria significativamente a experiência do usuário.

Autenticação e Autorização: Não há controle de acesso. A adição de um módulo de login/senha ou chaves de acesso seria crucial para um ambiente de produção, garantindo que apenas usuários autorizados possam enviar ou receber arquivos.

Suporte a Múltiplos Arquivos: O cliente só pode enviar um arquivo por execução. Permitir o envio de múltiplos arquivos em uma única sessão de conexão aumentaria a eficiência.

5. DIFICULDADES ENCONTRADAS NO DESENVOLVIMENTO

O desenvolvimento do projeto apresentou desafios típicos da programação de rede e concorrência

1. Sincronização e *Threading*: A principal dificuldade foi garantir que o servidor pudesse lidar com múltiplas conexões simultaneamente sem falhas. O uso de threads resolveu a concorrência, mas exigiu atenção especial ao ciclo de vida das conexões e ao fechamento dos sockets para evitar vazamentos de recursos.

2. Protocolo de Aplicação: A definição do protocolo de handshake (verificação de existência e envio de metadados) foi crucial. Tivemos que garantir que o servidor recebesse os metadados (nome e tamanho) de forma confiável e em um tamanho fixo (METADATA_SIZE) antes de iniciar a transferência binária do arquivo.

3. Tratamento de Sockets Bloqueantes: Gerenciar as chamadas recv() e sendall() de forma a garantir que todos os bytes fossem transferidos, especialmente com o uso de BUFFER_SIZE , exigiu testes rigorosos para evitar que a transferência fosse encerrada prematuramente.