CESAR SCHOOL
CIÊNCIAS DA COMPUTAÇÃO

CHAT DISTRIBUÍDO COM CONFIABILIDADE E CONCORRÊNCIA

RAFAEL ANDRADE LEITE BARROS
MIGUEL DE SOUZA BATISTA
TIAGO GURGEL AMORIM FERREIRA DE ABREU
VINICIUS DINIZ AMORIM SIMÕES

RECIFE
2025

SUMÁRIO

RESUMO

Apresentamos uma aplicação cliente-servidor que implementa comunicação confiável com fragmentação e confirmação (ACK/NACK), suporte a múltiplos clientes simultâneos e broadcast das mensagens entre clientes. O servidor utiliza threads por cliente e um processo separado para logging (multiprocessing), ilustrando concorrência, paralelismo e comunicação entre processos. O protocolo inclui handshake de três vias, cabeçalho binário com checksum e janela deslizante.

ABSTRACT

We present a client-server application that implements reliable communication with fragmentation and acknowledgment (ACK/NACK), support for multiple simultaneous clients, and message broadcasting among clients. The server uses a thread per client and a separate process for logging (multiprocessing), illustrating concurrency, parallelism, and inter-process communication. The protocol includes a three-way handshake, binary header with checksum, and sliding window.

1. INTRODUÇÃO

Sistemas distribuídos exigem mecanismos de comunicação confiável para mitigar perdas, corrupção e atrasos, e este projeto foi concebido com foco pedagógico para expor de forma transparente cada etapa desse processo: construímos sobre TCP um protocolo de aplicação enxuto com handshake de três vias, framing explícito com cabeçalho binário com tamanho, tipo, sequência, checksum e marcador de último fragmento, fragmentação controlada e confirmação positiva/negativa também conhecidos como ACK/NACK por fragmento, permitindo observar reconstituição e validação de mensagens em tempo real. A arquitetura ilustra simultaneamente concorrência, threads por cliente e thread receptora no cliente para evitar bloqueio de entrada, paralelismo, processo separado de logging via multiprocessing, e comunicação entre processos/threads, além de sincronização com Lock e Event para garantir shutdown limpo. O servidor centraliza sessões, realiza broadcast confiável após validar a integridade da mensagem completa e oferece operações auxiliares como a definição de apelido e a listagem de clientes que demonstram a extensibilidade do protocolo.

2. METODOLOGIA

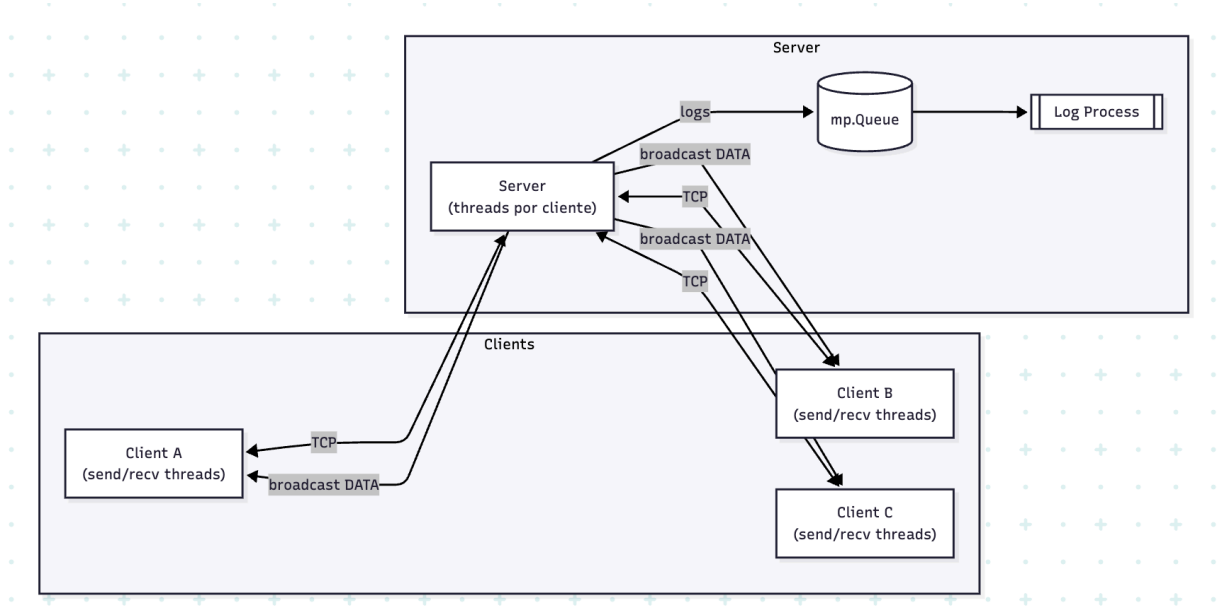
Para atingir o objetivo de desenvolver uma solução distribuída funcional, foi empregada uma metodologia de implementação baseada na arquitetura Cliente-Servidor sobre o protocolo TCP. A solução foi desenvolvida na linguagem Python, utilizando suas bibliotecas padrão para comunicação em rede, concorrência e paralelismo. A seguir, detalha-se a arquitetura e os conceitos aplicados.

2.1 ARQUITETURA DO SISTEMA

O sistema opera sobre uma arquitetura Cliente-Servidor centralizada. O Servidor, visto no arquivo server.py, atua como o ponto central de comunicação, responsável por aceitar conexões de múltiplos clientes, gerenciar sessões e retransmitir, fazer o broadcast, as mensagens recebidas para todos os outros participantes conectados. Cada Cliente, visto no arquivo client.py, é uma aplicação de terminal que estabelece uma conexão com o servidor, permitindo ao usuário enviar e receber mensagens em tempo real.

A comunicação é fundamentada na biblioteca socket, que provê a base para a troca de dados em rede. A Figura 1 ilustra a interação entre os componentes centrais do sistema.

Figura 1 – Diagrama da arquitetura distribuída do sistema



Fonte: Os autores, 2025

2.2 PROTOCOLO DE COMUNICAÇÃO CUSTOMIZADO

Sobre o TCP, foi implementado um protocolo de aplicação para estruturar a comunicação. Cada pacote trocado entre cliente e servidor possui um cabeçalho customizado de 11 bytes, contendo: o tamanho do payload, o tipo da mensagem, sendo ela por exemplo SYN, ACK ou DATA, , um número de sequência, um checksum para verificação de integridade e um flag para indicar o último fragmento de uma mensagem. Esse protocolo rege

todas as interações, desde o estabelecimento da conexão através de um three-way handshake (SYN, SYN-ACK, ACK) até a troca de dados e o encerramento da sessão.

2.3 IMPLEMENTAÇÃO DA CONCORRÊNCIA E PARALELISMO

A solução aplica os conceitos de concorrência e paralelismo de forma integrada. O servidor é multithreaded, criando uma nova thread (_client_worker em server.py) para cada cliente, o que permite o atendimento concorrente de múltiplas conexões. No lado do cliente, uma thread (_receiver_loop em client.py) gerencia o recebimento de mensagens de forma assíncrona. A consistência de dados compartilhados no servidor, como a lista de sessões (client_sessions), é garantida por um threading.Lock para evitar condições de corrida. Adicionalmente, o sistema explora o paralelismo ao delegar a tarefa de logging a um processo separado (_log_worker em server.py), utilizando uma multiprocessing.Queue para a Comunicação entre Processos (IPC), o que otimiza a performance ao isolar operações de I/O

3.RESULTADO E DISCUSSÃO

Nesta seção, são apresentados os resultados práticos obtidos com a execução do sistema de chat distribuído. A demonstração do funcionamento, dos testes de confiabilidade e das funcionalidades implementadas é consolidada através de um vídeo explicativo, complementado por capturas de tela que destacam os momentos cruciais da operação do software.

3.1 DEMONSTRAÇÃO COMPLETA COM VÍDEO

O funcionamento prático da aplicação é apresentado em um vídeo que demonstra uma sessão de uso real, documentando a sequência completa de operações desde a inicialização do servidor até a interação concorrente entre três clientes distintos. Na gravação, o servidor é iniciado e os clientes se conectam sequencialmente, com os logs evidenciando o sucesso do handshake de três vias para cada sessão. A robustez do protocolo é demonstrada quando um cliente envia uma mensagem longa, que é automaticamente fragmentada em múltiplos pacotes e depois remontada pelo servidor antes do broadcast, com cada fragmento sendo confirmado individualmente. Essa capacidade de broadcast valida o modelo de comunicação concorrente, onde a troca de mensagens entre os três clientes ocorre em tempo real. Por fim, o vídeo conclui com o uso dos comandos /nick e /who, exibindo a atualização dinâmica da lista de usuários conectados e a extensibilidade do protocolo. O vídeo pode ser acessado através do link a seguir:

Link para o vídeo: vídeo demonstração

3.2 DESTAQUES DA EXECUÇÃO

Este vídeo demonstra a aplicação no modo de interface de usuário interativa (--mode ui), focando na flexibilidade do sistema. A gravação exibe a capacidade do cliente de escolher entre os protocolos Go-Back-N e Selective Repeat antes da conexão, e em seguida, apresenta um cenário de comunicação com múltiplos clientes utilizando protocolos diferentes simultaneamente. Fica evidenciado que a funcionalidade de broadcast opera de forma transparente neste ambiente heterogêneo, validando a robustez do servidor em gerenciar diferentes configurações de sessão de forma concorrente.

Link para o vídeo: vídeo demonstração com UI

Este vídeo demonstra a robustez do protocolo de comunicação através de dois testes de falha simulados pela interface interativa. Primeiramente, no modo de corrupção de pacotes, os logs exibem o servidor detectando um erro de checksum e o cliente reenviando o pacote corrompido após receber um NACK. Na sequência, um teste de perda de pacotes mostra o mecanismo de timeout do cliente acionando uma retransmissão automática após não receber a confirmação (ACK) do servidor. A demonstração valida a capacidade do sistema de se recuperar autonomamente de erros e perdas na rede

Link para o vídeo: vídeo demonstração da confiabilidade

3.3 DISCUSSÃO DOS RESULTADOS

Os resultados apresentados, consolidados nos vídeos de demonstração, validam empiricamente o sucesso do projeto em atingir seus objetivos centrais. A primeira demonstração comprova a funcionalidade da arquitetura cliente-servidor e a correta implementação das características do protocolo, como a fragmentação de mensagens e o gerenciamento concorrente de múltiplos usuários via broadcast. A flexibilidade do sistema é evidenciada no segundo vídeo, que destaca não apenas a interface interativa, mas principalmente a robustez do servidor em gerenciar uma sessão heterogênea com clientes operando sob protocolos distintos (GBN e SR) de forma transparente. O ponto mais crítico, a confiabilidade, é comprovado no terceiro vídeo, onde os testes de falha demonstram a capacidade do sistema de se recuperar autonomamente tanto da corrupção de pacotes, através da verificação de checksum e do uso de NACKs, quanto da perda de pacotes, por meio do mecanismo de timeout e retransmissão. Em conjunto, essas demonstrações confirmam que o projeto traduziu com êxito os conceitos teóricos de sistemas distribuídos, concorrência e paralelismo em uma aplicação prática, funcional e resiliente.

Link para testar e ter seus próprios resultados: Link para o ReadMe

4. CONCLUSÃO

Este projeto alcançou com sucesso o objetivo de implementar um sistema de chat distribuído, aplicando de forma prática os conceitos de concorrência, paralelismo e sincronização. A solução funcional utiliza uma arquitetura cliente-servidor com um servidor multithreaded para atender múltiplos clientes, threading.Lock para garantir a consistência de dados, e um processo dedicado para logging com comunicação via mp.Queue para otimizar o desempenho. O trabalho demonstrou a viabilidade da arquitetura e a importância da aplicação correta desses conceitos para a criação de sistemas distribuídos eficientes. Como trabalhos futuros, sugere-se a implementação de criptografia e o desenvolvimento de uma interface gráfica.