# Relatório Técnico — Sistema P2P com Tracker Centralizado

Alunas: Gessica Silva e Luana da Ros

## 1 Introdução

Este relatório apresenta a arquitetura, as decisões de projeto e o funcionamento do sistema P2P desenvolvido, composto por um *Tracker* centralizado e múltiplos *Peers* capazes de registrar arquivos, consultar fontes, transferir blocos e reconstruir arquivos recebidos. A proposta central do projeto é implementar, de forma simplificada, um sistema de distribuição de arquivos inspirado em redes P2P clássicas, utilizando divisão em blocos, comunicação via sockets e controle concorrente por *threads*.

Além da arquitetura básica, foram implementadas duas extensões relevantes solicitadas na disciplina: (1) mecanismos de **tratamento de erros e estabilidade** e (2) **testes de desempenho e estresse**, incluindo coleta automática de métricas.

O foco deste documento é descrever as escolhas estruturais e as soluções adotadas, evidenciando o aprendizado prático sobre redes, concorrência e manipulação de arquivos binários.

# 2 Arquitetura Geral

A arquitetura adotada segue o modelo **Tracker + Peers**, no qual:

- O **Tracker** age como um servidor central responsável por:
  - registrar peers,
  - manter lista de arquivos presentes na rede,
  - responder consultas sobre quais peers possuem determinado arquivo,
  - atualizar periodicamente a lista de arquivos dos peers,
  - detectar falhas de resposta em consultas de verificação.
- Os **Peers** funcionam simultaneamente como clientes e servidores:
  - enviam requisições ao Tracker,
  - disponibilizam seus arquivos a outros peers,
  - transferem arquivos em blocos,
  - reconstruem arquivos recebidos,
  - registram métricas de desempenho em operações de download.

Essa abordagem foi escolhida por equilibrar simplicidade de implementação com a possibilidade de simular características reais de redes P2P.

## 2.1 Fluxo Geral do Sistema

1. O Peer inicia, lê seus arquivos locais e registra-se no Tracker.
2. O Tracker armazena o mapeamento: `peer_id → arquivos`.
3. Ao solicitar um arquivo:

(a) o Peer consulta o Tracker via `WHO_HAS filename`;

(b) recebe a lista de peers disponíveis;

(c) cria múltiplas threads para baixar o arquivo em paralelo;

(d) reconstrói o arquivo localmente;

(e) registra métricas da operação em um arquivo CSV.

# 3 Estrutura e Organização dos Arquivos

Para manipulação eficiente de dados binários, empregou-se a classe `FILES`, responsável por:

- dividir arquivos em blocos de 4096 bytes,
- manter metadados locais (tamanho, número de blocos),
- reconstruir arquivos recebidos a partir de blocos fora de ordem,
- salvar arquivos reconstruídos no sistema de arquivos.

Essa granularidade possibilita:

- transmissão paralela entre múltiplos peers,
- resiliência a ordem variável de chegada dos blocos,
- flexibilidade para futuras extensões (como hashes por bloco).

# 4 Implementação do Tracker

O Tracker foi projetado de maneira minimalista, mas com responsabilidade centralizadora suficiente para coordenar a rede.

## 4.1 Simplicidade e Confiabilidade

Ele mantém apenas:

- endereço IP e porta do peer;
- lista de arquivos;
- estado atualizado a cada consulta periódica.

## 4.2 Atualizações Periódicas e Estabilidade

Uma thread dedicada executa verificações frequentes enviando `VERIFY_FILES` para cada peer. Durante essa fase foram incorporados mecanismos de:

- detecção de peers que não respondem dentro do tempo limite;
- correção automática da lista de arquivos quando divergências são detectadas;
- registros de erros para auxiliar no diagnóstico.

Esses mecanismos atendem ao requisito de **estabilidade**, tornando o Tracker mais robusto diante de falhas individuais de peers.

## 4.3 Mensagens Suportadas

- `REGISTER`
- `WHO_HAS`
- `NEW_FILE`
- `DISCONNECT`
- `VERIFY_FILES`

# 5 Implementação do Peer

Os Peers atuam de forma híbrida:

- como servidor (respondendo `GET filename`);
- como cliente (consultando Tracker e outros peers).

## 5.1 Tratamento de Erros e Robustez

Para atender às exigências de **estabilidade**, foram adicionados mecanismos de proteção envolvendo:

- **timeouts** em conexões TCP;
- detecção de conexões recusadas ou quebradas;
- validação da integridade do cabeçalho de blocos;
- mensagens claras de erro ao usuário;
- tratamento isolado por thread para que falhas em um peer não afetem o restante do download.

Essas proteções tornaram o sistema capaz de se manter operacional mesmo quando peers se desconectam inesperadamente ou enviam dados incompletos.

## 5.2 Download Paralelo Estruturado

Cada peer detentor do arquivo é acessado simultaneamente. Uma `thread` dedicada executa:

1. conexão ao peer remoto;
2. recepção de metadados do arquivo (número de blocos e tamanho total);
3. recebimento de blocos em sequência;
4. escrita concorrente dos blocos em estrutura compartilhada protegida por *locks*.

Isso implementa efetivamente um modelo de **download paralelo**, em que cada peer contribui para completar o arquivo final.

# 6 Testes de Desempenho e Estabilidade

Atendendo aos requisitos da disciplina, o sistema incorpora dois módulos de avaliação: **benchmark** e **stress test**.

## 6.1 Benchmark de Download

O comando:

```
bench <filename> [runs]
```

repete múltiplas vezes o processo de download, registrando:

- nome do arquivo;
- tamanho em bytes;
- número de peers fornecedores;
- tempo total da operação.

Os resultados são gravados automaticamente em `download_times.csv`, com geração de cabeçalho quando o arquivo ainda não existe.

## 6.2 Teste de Estresse

O comando:

```
stress <filename> [n_threads]
```

abre diversas requisições simultâneas para o mesmo arquivo, avaliando:

- comportamento sob alta concorrência;
- estabilidade das conexões;
- impacto na latência de respostas;
- resiliência na reconstrução dos blocos.

Esse experimento permite observar o limite operacional do peer enquanto servidor.

## 6.3 Visualização dos Resultados

Um script auxiliar gera gráficos relacionando:

- número de peers fornecedores $\times$ tempo de download;
- variação de desempenho entre execuções sucessivas;
- influência do paralelismo na redução de latência.

Embora simples, esse módulo fornece uma avaliação empírica dos ganhos obtidos com a arquitetura paralela adotada.

# 7 Decisões de Projeto

As principais escolhas técnicas que definiram a arquitetura foram:

- **Modelo Tracker-centralizado**
- **TCP** para transmissão confiável
- **Blocos de 4096 bytes**
- **Paralelismo via threads**
- **Estruturas protegidas por locks**
- **Coleta automática de métricas para análise**

Essas decisões resultaram em um design modular, robusto e adequado para experimentação.

# 8 Limitações e Melhorias Futuras

- Tracker como *single point of failure*;
- Ausência de hashes de blocos para garantir integridade;
- Falta de detecção de corrupção de dados;
- Arquitetura não descentralizada.

Melhorias possíveis incluem:

- identificação por hashes (SHA-256);
- verificação de integridade por bloco;

- suporte a redundância de trackers;
- implementação de algoritmos de seleção inteligente de peers.

# 9 Conclusão

O projeto implementa com sucesso um sistema P2P funcional, modular e capaz de operar de forma concorrente e estável. A inclusão de mecanismos de erro e testes de desempenho aprofundou significativamente a robustez do sistema e permitiu realizar análises empíricas do comportamento da arquitetura sob diferentes configurações.

O resultado final é uma plataforma didática completa que demonstra conceitos essenciais de redes de computadores, concorrência, protocolos simples de comunicação e coordenação distribuída.