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*.

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.

- 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.

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. Quando um Peer deseja baixar um arquivo:
   (a) pergunta ao Tracker via `WHO_HAS filename`;
   (b) recebe a lista de peers com o arquivo;
   (c) abre múltiplas conexões paralelas para baixar blocos;
   (d) reconstrói o arquivo localmente.

3 Estrutura e Organização dos Arquivos

Para manipular arquivos de forma eficiente e permitir downloads concorrentes, implementou-se a classe `FILES`, responsável por:

- ler o arquivo original em blocos fixos de 4096 bytes;
- armazenar cada bloco com índice e conteúdo;
- reconstruir arquivos recebidos via bloco;
- salvar o arquivo completo no diretório do peer.

Essa divisão em blocos permite:

- paralelizar o download,
- distribuir a carga entre peers,
- reconstruir arquivos independentemente da ordem de recebimento.

4 Implementação do Tracker

O Tracker é um servidor TCP simples, cuja função é **coordenar** a rede, e não transferir dados. Seu projeto foi guiado pelos seguintes princípios:

4.1 Simplicidade e Confiabilidade

O Tracker mantém apenas:

- IP do peer,
- porta de serviço,
- lista de arquivos reportados.

4.2 Atualizações Periódicas

Uma *thread* dedicada executa verificações periódicas:

- solicita aos peers que confirmem sua lista de arquivos,
- atualiza automaticamente mudanças reais.

Esse mecanismo — embora simples — permite simular características reais de manutenção de estado em sistemas distribuídos.

4.3 Mensagens Suportadas

- `REGISTER` — registra novo peer;
- `WHO_HAS` — retorna peers que possuem um arquivo;
- `NEW_FILE` — notifica adição de arquivos;
- `DISCONNECT` — remove peer da rede.

5 Implementação do Peer

Os Peers foram projetados para operar em **modo híbrido**:

- **Servidor:** aguardando requisições GET de outros peers.
- **Cliente:** requisitando arquivos ao Tracker ou a outros peers.

5.1 Leitura Inicial de Arquivos

No momento da inicialização, o Peer varre seu diretório `peer_id/files` e transforma cada arquivo em um objeto `FILES`. Assim, o peer passa a conhecer:

- nome,
- quantidade de blocos,
- tamanho em bytes.

5.2 Servidor de Blocos

Ao receber um comando `GET filename`, o Peer:

1. localiza o arquivo,
2. envia bloco por bloco,
3. cada bloco é empaquetado com um cabeçalho contendo:
    - índice,
    - tamanho.

5.3 Download Paralelo

Ao solicitar um arquivo, o peer:

1. consulta o Tracker;
2. cria uma *thread* para cada peer que possui o arquivo;

3. recebe blocos simultaneamente;

4. sincroniza a escrita dos blocos em um dicionário compartilhado.

O uso de `threading.Lock` garante consistência durante a montagem dos blocos.

5.4 Reconstrução do Arquivo

Quando todos os blocos são recebidos, o peer chama:

```
read_from_blocklist → save_to_disk
```

realizando a reconstrução completa.

6 Decisões de Projeto

As principais escolhas técnicas que definiram a arquitetura foram:

- **Modelo Tracker-Centralizado:** Simplifica a coordenação geral e facilita depuração.
- **TCP como protocolo de transporte:** Necessário para garantir entrega ordenada e confiável de blocos.
- **Blocos de 4096 bytes:** Tamanho escolhido por equilíbrio entre velocidade, controle de memória e simplicidade.
- **Paralelismo via Threads:** Uma thread por peer detentor do arquivo, permitindo paralelização real do download.

- **Armazenamento Temporário em Memória:** Os blocos são agrupados em um dicionário antes da reconstrução, facilitando sincronização.

Essas decisões resultaram em um design modular, relativamente simples e eficiente do ponto de vista didático.

7 Limitações e Melhorias Futuras

Apesar de funcional, o sistema possui limitações naturais de sua arquitetura:

- O Tracker representa um **ponto único de falha**.
- Arquivos são identificados apenas pelo nome.
- Não há verificação de integridade (hash por bloco).
- Os peers assumem que todos os outros peers estão ativos.

Um ponto específico importante é o conflito entre arquivos diferentes com o mesmo nome. O uso de **hashes únicos** (como SHA-256) resolveria esse problema ao permitir identificação inequívoca de cada arquivo.

Essas melhorias são conhecidas, mas não foram implementadas devido à complexidade adicional necessária na estrutura de mensagens, sincronização e reescrita do protocolo.

8 Conclusão

O projeto implementa com sucesso um sistema P2P simples, funcional e modular, capaz de registrar peers, localizar arquivos na rede, transferi-los

em blocos e reconstruí-los de forma correta. A arquitetura escolhida permitiu equilibrar clareza, concisão e aplicabilidade prática, funcionando como uma excelente base para futuras extensões, como validação de integridade, hashing e modelos descentralizados.