Relatório

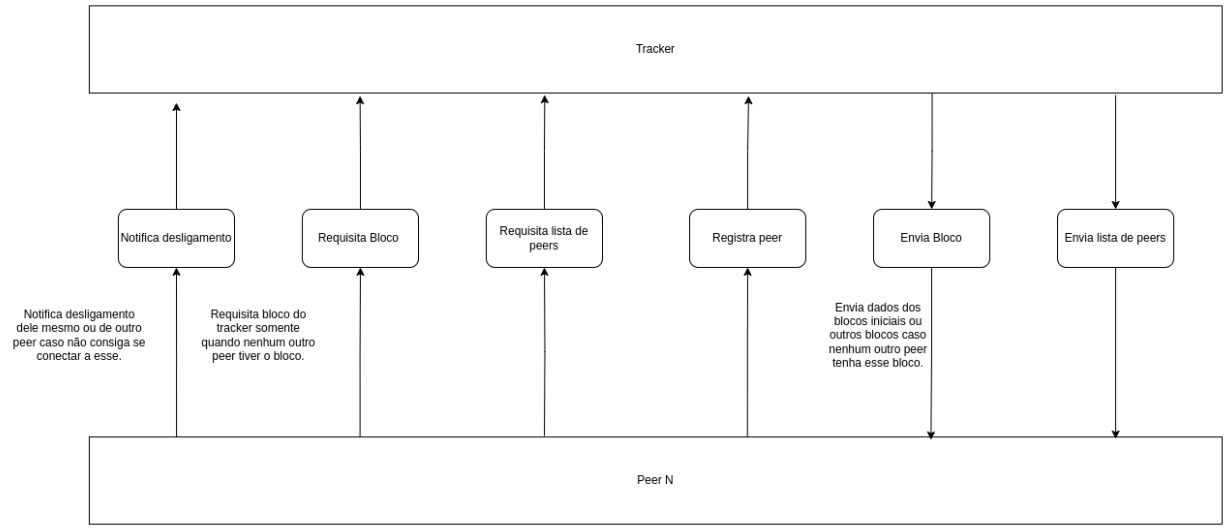
A arquitetura do sistema é composta por um Tracker central e múltiplos Peers. O Tracker é responsável por gerenciar a lista de Peers conectados, enquanto os Peers são os participantes da rede que baixam e compartilham os blocos do arquivo uns com os outros.

O Tracker pode servir como Peer, mas somente no caso onde nenhum Peer tiver um bloco. O Tracker também serve como seed inicial, pois ele contém o arquivo completo.

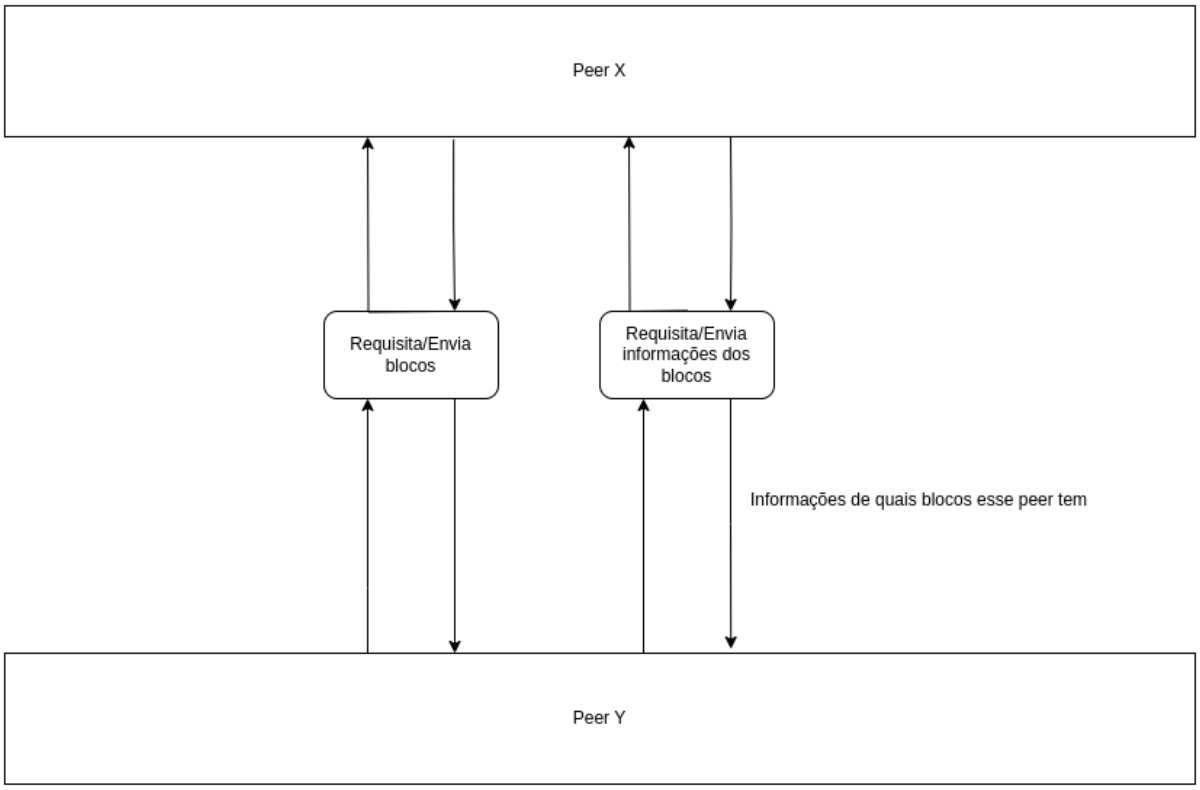
Diagrama da Arquitetura

Considerando o contexto loca em que tracker e peers estão inseridos.

Relação Tracker x Peer:



Relação Peer x Peer:



Descrição do Protocolo de Comunicação

O protocolo de comunicação é baseado em mensagens JSON trocadas via sockets TCP. As mensagens contêm um campo `action` que indica o tipo de operação, além de outros campos relevantes para cada ação.

Entidades:

- **Tracker:** Servidor central que mantém o registro dos peers e a disponibilidade de blocos.
- **Peer:** Cliente que baixa e compartilha blocos do arquivo.

Estados do Peer:

Os peers passam por diferentes estados durante a operação, embora não sejam explicitamente modelados como estados discretos no código, as ações implicam nos seguintes comportamentos:

- **Inicialização:** O peer se conecta ao tracker e registra sua presença, recebendo blocos iniciais e uma lista de peers conhecidos.

- **Descoberta de Peers:** Periodicamente, o peer solicita ao tracker uma lista atualizada de outros peers na rede.
- **Troca de Informações de Blocos:** Os peers solicitam e compartilham informações sobre os blocos que possuem com outros peers.
- **Seleção de Blocos (Rare-First):** O peer identifica os blocos mais raros (aqueles que menos peers possuem e que ele ainda não tem) para priorizar o download.
- **Seleção de Peers (Tit-for-Tat e Otimismo):** O peer seleciona os peers com os quais irá trocar dados, priorizando aqueles que fornecem blocos mais raros (Tit-for-Tat) e um peer "otimista" para explorar novas conexões.
- **Requisição e Download de Blocos:** O peer requisita blocos a outros peers ou ao tracker (somente caso nenhum outro peer tenha o bloco).
- **Anúncio de Bloco:** Após adquirir um novo bloco, o peer anuncia aos peers conhecidos que agora possui esse bloco.
- **Reconstrução do Arquivo:** Quando todos os blocos são baixados, o peer reconstrói o arquivo completo.
- **Desligamento:** O peer informa ao tracker que está saindo da rede.

Mensagens Trocadas:

As mensagens são objetos JSON com um campo `action` principal e outros campos dependendo da ação.

1. **Registro de Peer (`action: "register"`):**
    - **Remetente:** Peer
    - **Destinatário:** Tracker
    - **Conteúdo:**
        - `peer_id` (string): Identificador único do peer.
        - `listen_port` (int): Porta em que o peer está escutando por conexões de outros peers.
    - **Resposta do Tracker:**
        - `status` (string): "success" ou "error".
        - `initial_blocks` (list of int): Lista de índices dos blocos que o tracker designou inicialmente ao peer.

- `peers` (list of objects): Lista de outros peers já conectados, contendo `peer_id`, `ip` e `port`.

2. **Obtenção de Peers (`action: "get_peers"`):**
   - **Remetente:** Peer
   - **Destinatário:** Tracker
   - **Conteúdo:**
     - `peer_id` (string): Identificador do peer que está solicitando a lista.
   - **Resposta do Tracker:**
     - `status` (string): "success" ou "error".
     - `peers` (list of objects): Uma lista de peers ativos (até 5 peers selecionados aleatoriamente se houver mais de 5, caso contrário, todos), contendo `peer_id`, `ip` e `port`.
3. **Requisição de Bloco (Peer para Peer) (`action: "request_block"`):**
   - **Remetente:** Peer
   - **Destinatário:** Outro Peer
   - **Conteúdo:**
     - `sender_id` (string): Identificador do peer solicitante.
     - `block_index` (int): Índice do bloco desejado.
   - **Resposta do Peer (Destinatário):**
     - `status` (string): "success" ou "error".
     - `block_index` (int): Índice do bloco.
     - `block_data` (string): Dados do bloco (codificados, se necessário).
     - `reason` (string, opcional): Mensagem de erro se `status` for "error" (ex: "Choked or block unavailable").
4. **Requisição de Bloco (Peer para Tracker) (`action: "request_block_tracker"`):**
   - **Remetente:** Peer
   - **Destinatário:** Tracker
   - **Conteúdo:**

    - `sender_id` (string): Identificador do peer solicitante.
    - `block_index` (int): Índice do bloco desejado.
  - **Resposta do Tracker:**
    - `status` (string): "success" ou "error".
    - `block_index` (int): Índice do bloco.
    - `block_data` (string): Dados do bloco.
    - `reason` (string, opcional): Mensagem de erro se `status` for "error" (ex: "Bloco não disponível no tracker").
5. **Informação de Blocos Possuídos (`action: "have_blocks_info"`):**
  - **Remetente:** Peer
  - **Destinatário:** Outro Peer
  - **Conteúdo:**
    - `sender_id` (string): Identificador do peer que está enviando suas informações de blocos.
  - **Resposta do Peer (Destinatário):**
    - `status` (string): "success" ou "error".
    - `peer_id` (string): ID do peer que está respondendo.
    - `blocks_info` (list of boolean): Uma lista booleana indicando quais blocos o peer possui (True) ou não (False).
6. **Anúncio de Bloco (`action: "announce_block"`):**
  - **Remetente:** Peer
  - **Destinatário:** Outro Peer
  - **Conteúdo:**
    - `sender_id` (string): Identificador do peer que acabou de adquirir o bloco.
    - `block_index` (int): Índice do bloco recém-adquirido.
  - **Resposta do Peer (Destinatário):**
    - `status` (string): "success" (resposta simples de confirmação).
7. **Peer Offline (`action: "peer_offline"`):**

- **Remetente:** Peer (informando sobre si mesmo) ou Outro Peer (informando sobre um peer que falhou na conexão).
- **Destinatário:** Tracker
- **Conteúdo:**
  - `dead_peer_id` (string): Identificador do peer que está offline.
  - `sender_id` (string): Identificador do peer que está enviando a notificação.
- **Resposta do Tracker:**
  - `status` (string): "success" ou "error".

Mensagem	Remetente	Destinatário	Parâmetros Requisição	Parâmetros Resposta
Registro de Peer (register)	Peer	Tracker	peer_id, listen_port	status, initial_blocks, peers
Obtenção de Peers (get_peers)	Peer	Tracker	peer_id	status, peers
Requisição de Bloco (request_block)	Peer	Peer	sender_id, block_index	status, block_index, block_data, reason
Requisição de Bloco ao Tracker (request_block_tracker)	Peer	Tracker	sender_id, block_index	status, block_index, block_data, reason
Informação de Blocos (have_blocks_info)	Peer	Peer	sender_id	status, peer_id, blocks_info
Anúncio de Bloco (announce_block)	Peer	Peer	sender_id, block_index	status
Peer Offline (peer_offline)	Peer	Tracker	dead_peer_id, sender_id	status

Estratégias de Seleção de Blocos e Peers

No desenvolvimento deste sistema P2P, foram implementadas duas estratégias fundamentais para otimizar a distribuição e aquisição de blocos de arquivos: "Rarest

First" (blocos mais raros primeiro) e "Tit-for-Tat" (olho por olho).

Rarest First (Blocos Mais Raros Primeiro)

A estratégia "Rarest First" é empregada pelos peers para determinar qual bloco solicitar em seguida, priorizando a aquisição de blocos que são menos comuns na rede de peers conhecidos.

Implementação e Estruturas de Dados:

- `PeerSocket.blocks_owned`: Um array booleano (`[False] * TOTAL_FILE_BLOCKS`) mantido por cada peer, indicando quais blocos ele já possui.
- `PeerSocket.peer_blocks`: Um dicionário (`{peer_id: [bool, bool, ...]}`) que armazena as informações de blocos possuídos por cada peer conhecido. Esta estrutura é populada através de requisições `have_blocks_info` enviadas a outros peers.
- `PeerSocket.get_rarest_blocks()`: Este método itera sobre `TOTAL_FILE_BLOCKS` e, para cada bloco não possuído localmente (verificado em `self.blocks_owned[idx]`), conta quantos peers conhecidos (verificado em `self.peer_blocks.values()`) possuem esse bloco. Os blocos são então retornados em uma lista ordenada do mais raro (menor contagem) para o mais comum. Blocos que ninguém possui recebem um valor alto para garantir sua prioridade.

Funcionamento: A lógica principal reside no loop de download do método `PeerSocket.run()`, onde `get_rarest_blocks()` é invocado para obter a lista priorizada de blocos a serem requisitados.

Tit-for-Tat (Olho por Olho)

A estratégia "Tit-for-Tat" é utilizada para decidir quais peers "desengasgar" (unchoke), ou seja, para quais peers o peer atual permitirá o envio de blocos. Esta estratégia é baseada na reciprocidade, incentivando a cooperação e o compartilhamento de recursos.

Implementação e Estruturas de Dados:

- `PeerSocket.unchoked_peers`: Um `set` que armazena os `peer_id`s dos peers que estão atualmente "desengasgados" e, portanto, podem requisitar blocos do peer atual.
- `PeerSocket.tit_for_tat()`: Este método é executado periodicamente (a cada 10 segundos no `PeerSocket.run()`).

    - Ele primeiro calcula a "contribuição" de cada peer conhecido (presente em `self.peer_blocks`), avaliando quantos dos **blocos mais raros** (obtidos via `self.get_rarest_blocks()`) aquele peer possui.
    - Os peers são então ordenados com base nessa contribuição.
    - Quatro peers com a maior contribuição são selecionados como "fixos" e adicionados a `self.unchoked_peers`.
    - Adicionalmente, um peer é escolhido aleatoriamente dentre os "candidatos" (peers não selecionados como fixos) para ser o "peer otimista" e também é adicionado a `self.unchoked_peers`. O desengasgamento otimista garante a descoberta de novos peers e a prevenção de situações de deadlock.
- **`PeerSocket.handle_block_request()`**: Este método verifica se o `sender_id` da requisição está presente em `self.unchoked_peers` antes de enviar o bloco solicitado, aplicando a regra de "engasgamento"

Dificuldades Enfrentadas na Implementação

Tracker como seed inicial e se comportando como peer.

Um dos principais desafios durante o desenvolvimento foi a integração do tracker como um ponto de partida inicial para a distribuição dos blocos do arquivo. Inicialmente, a arquitetura foi pensada com os peers se comunicando apenas entre si para a troca de blocos. No entanto, para garantir que o sistema pudesse iniciar a distribuição de blocos mesmo sem nenhum peer com o arquivo completo, tornou-se necessário que o tracker também fosse capaz de atuar como um "seed" inicial.

Isso exigiu a criação de métodos específicos para que os peers pudessem solicitar blocos diretamente ao tracker. Além disso, a lógica de busca de blocos dentro da função `run` dos peers teve que ser adaptada. Agora, se um peer não conseguir encontrar um determinado bloco entre os outros peers conectados (seja porque nenhum deles o possui ou porque os peers que o possuem estão "choked"), ele tenta requisitar esse bloco diretamente ao tracker. Essa mudança foi crucial para a robustez do sistema, permitindo que o processo de download começasse e continuasse mesmo em cenários com poucos peers ou com todos os peers inicialmente sem o arquivo completo.

Implementação de envio código binário para suprir outros tipos de arquivo

Apesar dos avanços, a implementação do envio de código binário para suportar outros tipos de arquivo representou um obstáculo significativo que decidimos postergar. A complexidade de serializar e desserializar dados binários de forma genérica e eficiente, garantindo a integridade dos blocos e a compatibilidade entre diferentes sistemas operacionais, adicionaria uma camada de desafio considerável à arquitetura atual. Isso nos levou a focar o projeto exclusivamente em arquivos de texto, otimizando o tempo de desenvolvimento e a estabilidade da simulação. A ambição de suportar dados binários, embora reconhecida como essencial para um sistema BitTorrent completo, permanece como um aprimoramento futuro, dada a sua complexidade intrínseca.

Um bloco requisitado/enviado por vez

Outra limitação notável na abordagem atual é a estratégia de transferência de blocos um por um. Como o sistema envia e recebe apenas um bloco por vez, e as requisições são sequenciais, a velocidade total de download é diretamente atrelada à quantidade de blocos configurada, independentemente do número de peers disponíveis na rede local. Isso significa que, mesmo com muitos peers online, o tempo para baixar o arquivo completo será semelhante ao tempo necessário para processar cada um dos blocos individualmente. Para uma rede local, onde a latência é mínima, essa abordagem simplificada permite focar na lógica do protocolo; entretanto, em uma rede mais distribuída como a internet, a falta de paralelismo na transferência de blocos seria um gargalo significativo, onde fatores como latência e largura de banda teriam um impacto muito maior no desempenho geral do download.

Testes

Foram realizados testes para avaliar o desempenho do sistema sob diferentes configurações de blocos e número de peers. O teste consistiu em abrir um terminal para o tracker e um para cada peer separadamente. Os resultados detalhados são apresentados abaixo.

- 3 Peers
  - 20 blocos

Peer ID	Tempo (s)	Msg Enviadas	Msg Recebidas

peer1	55.11	41	77
peer2	55.11	43	79
peer3	55.11	44	48

Tempo Médio de Download por Peer: 55.11 segundos
Média de Mensagens Enviadas por Peer: 42
Média de Mensagens Recebidas por Peer: 68

- 100 blocos

Peer ID	Tempo (s)	Msg Enviadas	Msg Recebidas
peer1	295.57	233	429
peer2	295.60	235	323
peer3	295.62	236	270

Tempo Médio de Download por Peer: 295.60 segundos
Média de Mensagens Enviadas por Peer: 234
Média de Mensagens Recebidas por Peer: 340

- 200 blocos

Peer ID	Tempo (s)	Msg Enviadas	Msg Recebidas
peer1	596.20	473	997
peer2	596.29	475	588
peer3	596.34	475	559

Tempo Médio de Download por Peer: 596.28 segundos
Média de Mensagens Enviadas por Peer: 474
Média de Mensagens Recebidas por Peer: 714

- 5 Peers
  - 20 blocos

Peer ID	Tempo (s)	Msg Enviadas	Msg Recebidas
peer1	55.12	76	136
peer2	55.14	80	113
peer3	55.13	82	108
peer4	58.14	80	93

peer5	76.20	59	72

Tempo Médio de Download por Peer: 59.95 segundos
Média de Mensagens Enviadas por Peer: 75
Média de Mensagens Recebidas por Peer: 104

- 100 blocos

Peer ID	Tempo (s)	Msg Enviadas	Msg Recebidas
peer1	295.65	464	789
peer2	295.73	467	557
peer3	295.73	468	551
peer4	295.73	470	560
peer5	295.77	469	541

Tempo Médio de Download por Peer: 295.72 segundos
Média de Mensagens Enviadas por Peer: 467
Média de Mensagens Recebidas por Peer: 599

- 200 blocos

Peer ID	Tempo (s)	Msg Enviadas	Msg Recebidas
peer1	596.39	944	1450
peer2	596.45	947	1258
peer4	596.50	949	1100
peer3	599.57	947	1142
peer5	596.49	949	1089

Tempo Médio de Download por Peer: 597.08 segundos
Média de Mensagens Enviadas por Peer: 947
Média de Mensagens Recebidas por Peer: 1207

O tempo para completar o download do arquivo se mostra bastante consistente entre os peers em cada cenário, mesmo com variações no número de mensagens trocadas. Isso é particularmente relevante porque, nessa implementação, cada peer envia e recebe um bloco por vez, ou seja, uma requisição equivale a um bloco e os testes foram feitos em um ambiente local.

Reflexões

Diogo

“Este projeto se revelou uma jornada bastante esclarecedora, mergulhando a gente de cabeça nas complexidades dos sistemas distribuídos. Mais do que apenas codificar, fomos levados a refletir profundamente sobre a robustez da comunicação em ambientes imprevisíveis, onde a perda ou a desordem de "pacotes" é uma realidade constante. A decisão de focar em arquivos de texto, embora uma simplificação prática diante dos desafios de lidar com transferências binárias mais intrincadas, ressaltou a importância de priorizar a funcionalidade. E a inteligência da rede? Essa veio com a implementação dos algoritmos Rarest First e do Tit-for-Tat simplificado. Eles transformaram o que seria um simples sistema de transferência em um ambiente dinâmico, onde a colaboração é incentivada e a estratégia é chave para a eficiência do download, mostrando as diversas camadas de complexidade que precisam ser gerenciadas para que uma rede descentralizada opere de forma coesa e eficaz.”

Gabriella

“Com esse projeto tive a oportunidade de aprofundar conhecimentos de threads e sockets na prática, criando conexões, enviando e recebendo mensagens, além die lidar com comunicação de processos, sobretudo tratar erros quando a um processo (peer) não responde. No contexto de BitTorret foi interessante verificar o que fazer quando um peer simplesmente sair do circuito mesmo que ele não tenha completado o download. Um cenário real, muito comum de acontecer. Outra coisa que eu só tinha ouvido falar por cima e agora consegui entender melhor foi como funcionam os algoritmos distribuídos usados em redes P2P, como o BitTorrent. Mesmo que a gente tenha feito versões simplificadas e adaptadas, deu pra perceber o porquê de estratégias como o Tit-for-Tat e o Rarest First existirem.”