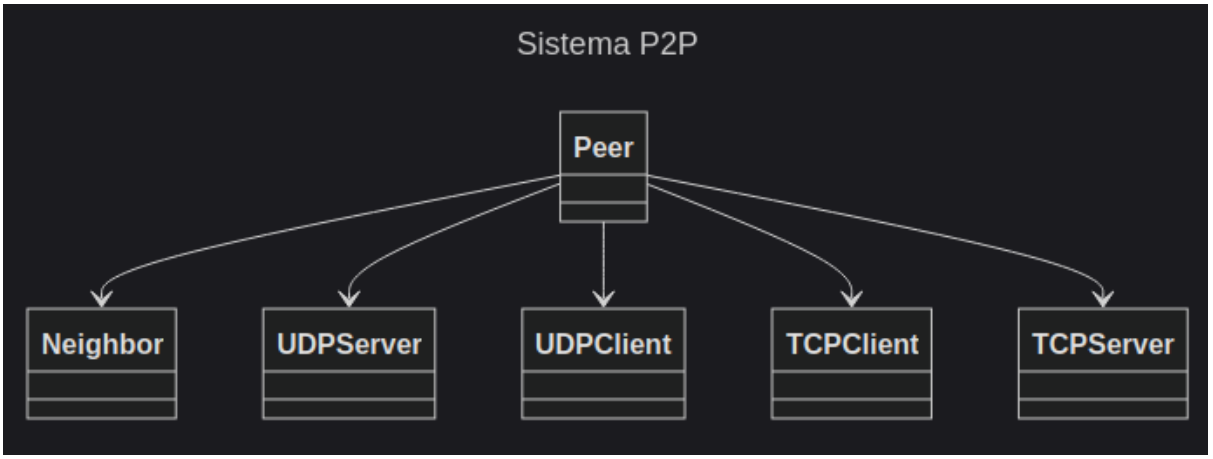
Sistema P2P

Disciplina: Computação Distribuída - INE5418
Nome do grupo: Grupo
Integrantes: Eduardo de Moraes (19203167) e Stephan Krug (21100141)

1. Estrutura

O sistema P2P foi estruturado de acordo com o seguinte esquema:



- A classe Peer é responsável por centralizar a lógica de busca de um arquivo, a partir do momento em que é requisitado pelo usuário. Ela pode ser realizada de duas formas: localmente - caso o peer tenha todos os chunks em seu sistema - ou pela rede - caso o peer não possua alguns/nenhum chunk;
- Cada peer possui um conjunto de Neighbors, que são apenas um encapsulamento das informações referentes aos vizinhos do peer: ID, endereço e porta UDP;
- A classe UDPClient é responsável por enviar uma mensagem UDP aos vizinhos do peer. Essa mensagem pode partir de duas fontes: um arquivo requisitado pelo usuário ou uma retransmissão proveniente de uma requisição de arquivo feita por outro peer;
- Cada peer possui um UDPServer, que é iniciado a partir do momento em que o peer inicia sua execução. Essa classe é responsável por realizar a busca no sistema de arquivos de um peer e localizar quais chunks o peer possui disponível, decrementar o TTL e pedir ao peer para retransmitir uma mensagem aos seus vizinhos, por um UDPClient;
- A classe TCPClient é responsável por receber arquivo(s) de um servidor TCP, que representam chunk(s) e/ou o arquivo completo que o usuário requisitou;
- A classe TCPServer é instanciada assim que um peer responde que possui um arquivo de interesse a outro peer. Após a instanciação, o servidor espera por conexões TCP de clientes, para então prover os arquivos de interesse a eles.

2. Funcionamento

2.1. Inicialização

Após o início da execução, a main do sistema realiza a criação de um Peer. Esse peer é responsável por:

1. Ler o arquivo de topologia, de forma a verificar quais são os seus vizinhos;
2. Ler o arquivo de configuração, de forma a verificar qual o endereço UDP, porta UDP e velocidade deste peer, além de qual o endereço UDP e porta UDP de cada um dos seus vizinhos;
3. Realizar a criação de objetos que encapsulam as informações referentes a cada um dos vizinhos;
4. Criar um UDPServer, que por sua vez fica esperando por uma mensagem UDP de busca de arquivo de outros peers.

Por fim, a main fica aguardando o usuário prover o nome de um arquivo de metadados que será consultado para realizar uma busca pelo arquivo na rede P2P.

2.2. Busca de Arquivo

Assim que o usuário prover o nome de um arquivo de metadados ao sistema, o peer realiza algumas verificações:

1. Se o arquivo de metadados não está presente no diretório do peer, a busca é abortada;
2. Se o arquivo especificado pelo arquivo de metadados já está presente localmente, a busca é abortada;
3. Se todos os chunks de um arquivo especificado pelo arquivo de metadados já estão presentes localmente, o arquivo completo é criado e a busca é abortada.

Caso nenhuma das verificações acima aborte a busca, o peer cria um buffer responsável por guardar a melhor possibilidade de busca para cada chunk e para o arquivo completo.

O formato de preenchimento do buffer é o seguinte:

```
[
   {
        'chunk': 0, # Número do chunk
        'address': '127.0.0.1', # Endereço TCP em que o chunk será
buscado
        'port': '4000', # Porta TCP em que o chunk será buscado
        'time': 60 # Estimativa em segundos para a realização da busca
pelo chunk
   },
   ...
]
```

Para os chunks que já estão presentes localmente, os campos address e port são preenchidos com ‘local’ e o campo time é preenchido com 0.

Exemplo: caso o arquivo a ser buscado tenha 7 chunks, o buffer é criado com 8 posições. As 7 primeiras posições representam a melhor possibilidade de busca de cada um dos 7 chunks e a última posição representa a melhor possibilidade de busca para o arquivo completo. Caso o peer já possua algum(ns) do(s) chunk(s) localmente, o buffer é inicializado com essas posições já preenchidas, em que o campo time é igual a 0.

Após realizar a criação de seu buffer, o peer cria um UDPClient responsável pelo flooding com a requisição por um arquivo (ver seção Requisição para descoberta de arquivos (flooding)). Além disso, o UDPClient fica bloqueado esperando receber mensagens de servidores UDP, contendo informações sobre o(s) chunk(s)/arquivo completo a ser(em) buscado(s) e o endereço de busca.

Cada servidor UDP que receber uma requisição por um arquivo realiza a busca localmente pelo arquivo completo ou por chunks daquele arquivo, cria um servidor TCP responsável por prover ao peer o(s) arquivo(s), decrementa o TTL e redireciona a mensagem aos seus vizinhos (por meio de outro UDPClient, criado exclusivamente para esse fim). Além disso, envia uma resposta com as informações de endereço de busca e quais os arquivos presentes no peer.

O UDPClient do peer que iniciou a busca espera receber requisições até que atinja seu período de timeout. Após isso acontecer, o peer verifica se o arquivo pode ser buscado integralmente e, caso não seja possível, aborta a busca. Caso seja possível, o peer escolhe a melhor técnica para a busca do arquivo: por chunk ou pelo arquivo completo. Isso é realizado somando o tempo de busca de cada chunk e comparando ao tempo de busca de um arquivo completo e escolhendo o menor entre os dois. Em caso de empate, a busca por chunks é escolhida, já que pode ser realizada paralelamente e dá a habilidade de o peer prover seus chunks a outros peers.

- Em caso de busca por chunk, são criados vários clientes TCPs (limitados a um número máximo de threads). Cada um se conecta ao servidor TCP em questão e requisita todos os chunks a serem retornados por aquele peer;
- Em caso de busca pelo arquivo completo, um cliente TCP é criado e se conecta ao servidor TCP, que proverá o arquivo completo.

Em ambos os casos, o servidor TCP envia uma mensagem especificando qual arquivo será enviado a seguir e, então, envia o conteúdo do arquivo. O cliente TCP cria o arquivo em um diretório tmp/ até que o recebimento do arquivo seja concluído, momento em que esse arquivo é movido para o diretório raiz do peer.

Assim que o peer recebe todos os arquivos de todos os servidores, ele realiza a criação do arquivo completo e permite que o usuário escolha o próximo arquivo a ser buscado.

2.3. Controle de velocidade

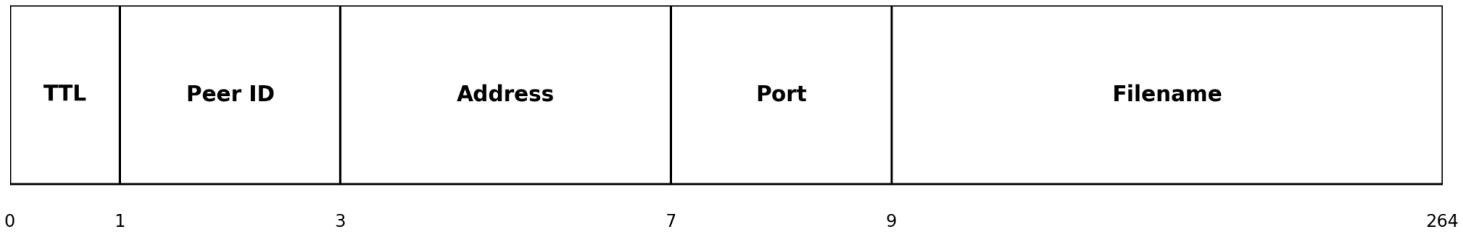
O controle de velocidade é realizado da seguinte forma:

- Assim que o servidor TCP de um peer se conecta a um cliente, ele envia uma mensagem ao peer informando que uma nova conexão foi realizada. O peer, que

possui uma variável com o número de conexões concomitantes, incrementa esse valor.
- A partir do momento em que um usuário requisita um arquivo, o peer realiza a estimativa de tempo para envio deste. Para isso, o peer divide o tamanho do arquivo pelo resultado da soma do número de conexões ativas naquele momento e 1.

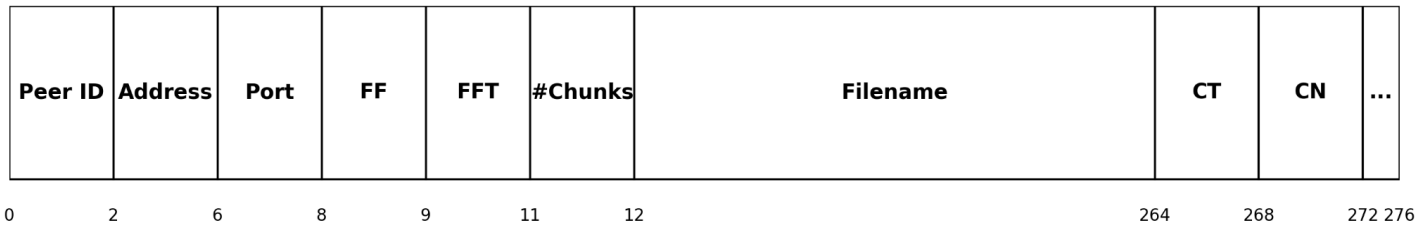
3. Formatos de Mensagens

3.1. Requisição para descoberta de arquivos (flooding)



Descrição dos campos:
- TTL é o número de saltos a serem realizados.
- Peer ID é o ID do peer que realizou a requisição do arquivo inicialmente.
- Address é o endereço UDP do peer que realizou a requisição do arquivo inicialmente.
- Port é a porta UDP do peer que realizou a requisição do arquivo inicialmente.
- Filename é o nome do arquivo que o peer requisitou.

3.2. Resposta da descoberta de arquivos



Descrição dos campos:
- Peer ID é o ID do peer que possui o arquivo.
- Address é o endereço TCP do servidor que proverá o arquivo.
- Port é a porta TCP do servidor que proverá o arquivo.
- FF (Full File) é um booleano que indica se o servidor possui o arquivo completo - não dividido em chunks.
- FFT (Full File Time) indica a estimativa de tempo para o servidor prover o arquivo completo.
- #Chunks (Number of Chunks) indica o número de chunks que o servidor possui.

- Filename indica o nome do arquivo que será provido.
- CT* (Chunk Time) indica a estimativa de tempo para prover o chunk.
- CN* (Chunk Number) indica o número do chunk.

*Os parâmetros CT e CN são opcionais e seu número de aparições segue o número indicado pelo parâmetro #Chunks.

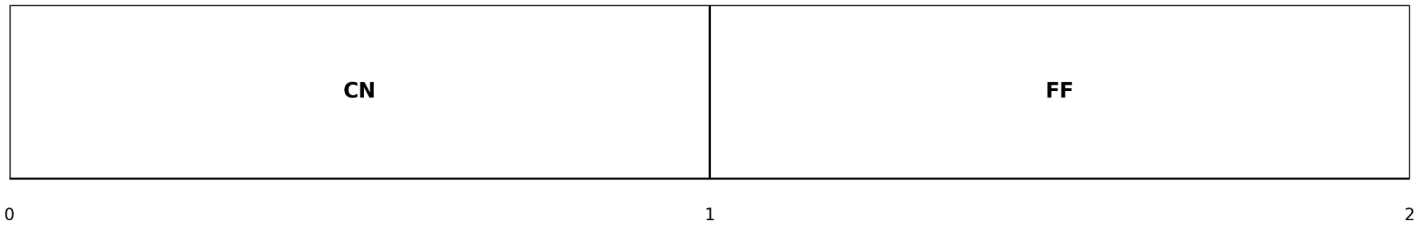
3.3. Requisição de envio de arquivo(s) ao servidor TCP

Descrição dos campos:

- #Chunks (Number of Chunks) indica o número de chunks a serem providos pelo servidor TCP. Caso seja 0, o servidor entende que deve prover o arquivo completo.
- Filename indica o nome do arquivo a ser buscado.
- CN* (Chunk number) indica o número do chunk a ser buscado.

*O parâmetro CN é opcional e seu número de aparições segue o número indicado pelo parâmetro #Chunks.

3.4. Especificação do arquivo a ser enviado pelo servidor TCP



Descrição dos campos:

- CN (Chunk Number) é o número do chunk a ser enviado. Em caso de envio do arquivo completo, será enviado 0.
- FF (Full File) é um booleano que indica se será enviado o arquivo completo ou não.

4. Execução

A especificação de como executar o código e informações complementares estão presentes no README.md do projeto.

4.1. Exemplo 1 - Busca de arquivo por chunks de diferentes peers

Este exemplo está indicado no diretório example/ do projeto.

Nele o peer 0 realizou a busca pelo arquivo image.png, que está dividido em 4 chunks da seguinte forma:

- Peer 1 possui o chunk 0.
- Peer 2 possui o chunk 1.
- Peer 3 possui os chunks 0 e 2.
- Peer 4 possui o chunk 3.

Neste caso, os chunks foram providos pelos peers 2, 3 e 4. O peer 3 foi escolhido para prover o chunk 0 pois, mesmo enviando 2 arquivos, ele possui uma velocidade estimada maior de envio de arquivos. Esse resultado poderia ser alterado caso outro peer tenha requisitado pelo mesmo chunk 0 enquanto o peer 3 já está provendo os chunks ao peer 0, por exemplo.

4.2. Exemplo 2 - Busca de arquivo completo de um peer

Este exemplo está indicado no diretório example2/ do projeto.

Nele o peer 0 realizou a busca pelo arquivo image.png, que está dividido em 4 chunks e 1 arquivo completo da seguinte forma:

- Peer 1 possui o chunk 0.
- Peer 2 possui o chunk 1.
- Peer 3 possui os chunks 0, 2 e o arquivo completo.
- Peer 4 possui o chunk 3.

Neste caso, como o tempo para prover o arquivo completo era menor caso buscado diretamente, ao invés de uma busca dividida para cada chunk, esta foi a estratégia escolhida. Esse resultado poderia ser alterado caso haja um aumento da velocidade de outros peers ou caso o peer 3 esteja muito congestionado com outras requisições.

5. Trabalhos futuros

Durante o desenvolvimento do trabalho notamos algumas melhorias que poderiam compor trabalhos futuros:

1. O peer que requisita um arquivo poderia ser modificado para guardar todos os possíveis peers que possuem um arquivo. Isso seria útil em 2 cenários:
    a. Caso um peer que foi escolhido e está provendo um arquivo seja desconectado, o peer que requisitou o arquivo pode escolher outro sem ter que repetir o flooding.

    b. Caso um peer tenha enviado uma estimativa e o envio do arquivo não esteja correspondendo à velocidade da estimativa, ele poderia escolher outro peer para enviar o restante do arquivo.
2. Permitir a busca paralela de um mesmo chunk em diferentes peers, caso essa estratégia seja mais rápida que um único peer provendo o arquivo. Para isso seria necessário enviar informações do tamanho de cada chunk e permitir que o peer realize a escolha considerando uma abordagem em paralelo para cada chunk.
3. Incrementar o timeout do UDPClient e o TTL para o arquivo caso uma requisição de um arquivo não tenha sido respondida com sucesso a tempo.
4. Enviar uma mensagem de “não escolha” aos peers não escolhidos, de modo que eles possam encerrar os servidores TCP e guardar recursos.