Peer-to-Peer File Sharing System

Team

Group 11

Asmitha Ramesh – UFID: 54996605

Sai Pande – UFID: 37696687

Introduction

This project implements a BitTorrent-inspired Peer-to-Peer (P2P) File Sharing System. It supports simultaneous uploading and downloading across peers by incorporating techniques such as file segmentation, choking/unchoking, bitfield exchange, and dynamic peer prioritization.

Project Overview

Each peer operates as both a client and a server, communicating over TCP sockets. Files are divided into smaller segments for efficient parallel transfer, enabling scalable and resilient decentralization sharing.

Demo Video

Watch the system demonstration: DEMO

https://uflorida-my.sharepoint.com/personal/saipande_ufl_edu/_layouts/15/stream.aspx?id=%2Fpersonal%2Fsaipande%5Fufl%5Fedu%2FDocuments%2FComputer%20Network%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview%2Ed93429b3%2D7b4a%2D406d%2D8e8d%2D911d480fb516

Core Mechanisms

Choking and Unchoking

Peers prioritize a limited number of neighbors based on their download rates. Others are choked to control bandwidth. Optimistic unchoking is used to randomly give new peers a chance to download.

Bitfield Management

Each peer maintains a bitfield representing the pieces it owns. After the handshake, bitfields are exchanged to determine which pieces are needed.

Request and Piece Exchange

Unchoked peers send `REQUEST` messages for missing pieces. A `PIECE` message with the data is returned, enabling incremental file completion.

Logging and File Segmentation

Events such as peer connections, piece transfers, and system milestones are logged. Files are split into chunks and reassembled after download.

Important Functions

- `refreshPreferredPeers()`: Selects top-performing peers based on download rate.
- `chooseOptimisticPeer()`: Randomly unchokes one choked peer.
- `hasPiece(int index)`: Checks if a peer owns a specific chunk.
- `mergeChunks()`: Merges all downloaded pieces into the complete file.

System Workflow

1. Peers initialize configuration using `Common.cfg` and `PeerInfo.cfg`.
2. Connections are established and identities verified using handshakes.
3. Bitfield exchange occurs.
4. Choking/unchoking manages bandwidth allocation.
5. File pieces are exchanged and downloaded.
6. Completion is logged after full download.

Conclusion

This system offers efficient and scalable peer-to-peer file sharing through decentralized control, dynamic prioritization, and modular communication. It simulates key aspects of real-world P2P protocols with clarity and performance awareness.