Dhirubhai Ambani
Institute of Information and Communication Technology

Distributed Systems (IT559)

PeerNet

A P2P Data Sharing System

Team Id: 13

202201163 – Jay Goyani
202201180 – Darshak Kukadiya
202201503 – Manoj Dhundhalva

1. Problem Statement :

Traditional client-server file-sharing systems have a big problem. If many people try to download the same file at the same time, the server slows down. To fix this, we will make a **peer-to-peer (P2P) file-sharing system using C++**. In this system, people can download files in parts (chunks) from different users instead of a single server. This makes downloading faster and reduces the load on one server.

2. System Overview :

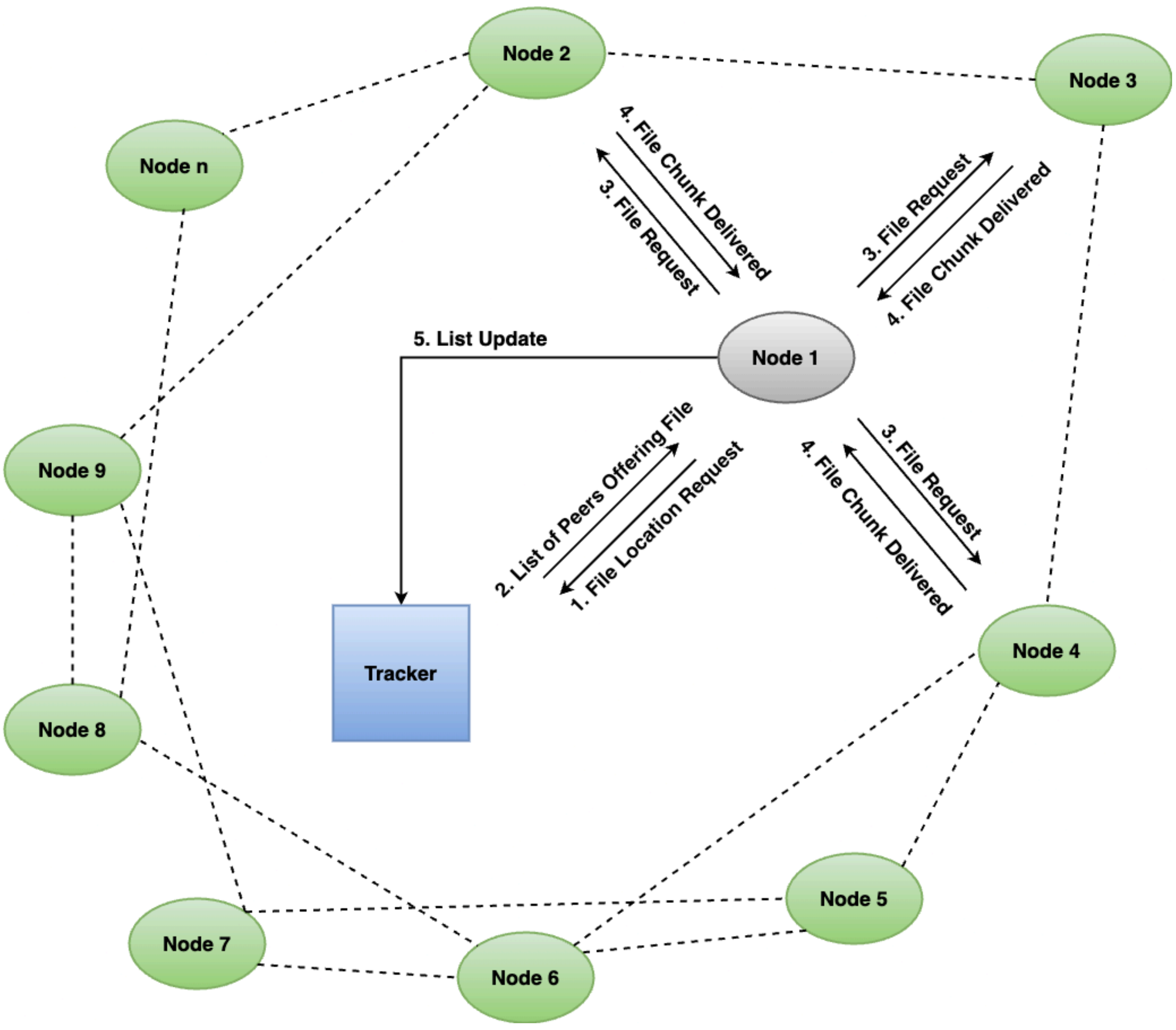
The proposed system will implement a **BitTorrent-based file-sharing network** with the following components:

1. **Tracker** – A centralized server responsible for maintaining a list of peers and the files they possess.
2. **Peers (Nodes)** – Users who participate in file sharing by downloading and uploading file chunks.

This system will demonstrate how P2P networks can efficiently distribute large files by splitting them into smaller chunks and allowing peers to exchange these chunks. By reducing dependency on a central server, the approach will enhance scalability and robustness in file distribution.

3. System Architecture and Components :

High-Level Architecture Diagram :



Components:

- **Tracker:**
  - Registers new users (peers).
  - Keeps a record of who has which file parts.
  - periodically checks peer activity to detect failures.
- **Peers (Nodes):**
  - Download and upload file chunks.
  - Connect with the tracker and other peers.

- **File Chunks:**
    - Large files are divided into smaller parts.
    - Peers exchange these chunks for faster downloads.

4. Distributed System Concepts Applied :

- **Peer-to-Peer (P2P) Networking:**
  The system operates on a peer-to-peer model where each node (peer) can act as both a client and a server. Instead of relying on a central server to serve files, peers directly exchange file chunks with one another, enabling decentralized resource sharing.

- **Decentralized Coordination with Centralized Discovery:**
  While the tracker plays a central role in helping peers discover each other by maintaining metadata about files and active nodes, the actual file transfer process is entirely decentralized. Once peers are discovered, they establish direct communication channels for chunk exchange, reducing load on the tracker.

- **Concurrency and Parallelism:**
  The system uses multi-threading to handle concurrent upload and download tasks. This allows peers to download multiple file chunks from different peers simultaneously, while also serving chunks to others, enhancing overall throughput and responsiveness.

- **Fault Tolerance at Peer Level:**
  To handle unreliable or inactive peers, each peer periodically sends heartbeat messages to the tracker. If a peer becomes unresponsive, the tracker detects the failure based on missing heartbeats and removes the peer from the active list. This helps maintain a clean and updated view of the network.

- **Chunk-Based Load Distribution:**
  Large files are divided into smaller chunks, and these chunks are distributed across multiple peers. During downloads, a requesting peer retrieves chunks from several sources in parallel. This naturally balances network load and speeds up file transfers.

- **Data Partitioning and Redundancy:**
  By breaking files into chunks and distributing them across the network, the system achieves both partitioning and a degree of redundancy. If one peer becomes unavailable, the same chunk may be available from another, improving reliability and availability.

5. Implementation Details :

1. **Programming Language**: The project is implemented in **C++**, leveraging both the language's standard library features (such as STL containers) and low-level system calls.
2. **Libraries**:
   a. **sys/socket:** For network communication.
   b. **pthread:** For concurrent operations and multithreading.
   c. **mutex:** For thread synchronization.
   d. **fstream**: For file I/O operations.
   e. **vector, map, set, etc.:** For managing data structures.
   f. **sys/stat, fcntl, sys/mman:** For file metadata, file control options, and memory-mapped file operations.
3. **APIs and Protocols:**
   - **UDP for Communication Protocol:** The project uses UDP for fast, connectionless message passing between nodes in the network.
   - **File and Directory APIs:** System calls such as `stat()`, `mkdir()`, and functions from `<dirent.h>` help in

managing file operations, checking file existence, reading directories, and more.

4. **Tools**:
    a. Git – Version control
5. **Platform Compatibility**: Linux, macOS

6. Challenges Faced and Solutions :

Throughout the development of our distributed P2P file-sharing system, we encountered multiple challenges. We managed to address some critical issues successfully, while others still represent opportunities for future refinement. Below is an outline of the main challenges along with our solutions and open issues:

Challenges We Successfully Addressed :

- **Consistency of Node Status:**

  Challenge: In a dynamic network, peers may join or leave unexpectedly. Keeping an accurate and up-to-date view of node availability was essential to ensure smooth file transfers.
  Solution: We implemented a robust heartbeat mechanism. Each peer periodically sends heartbeat messages to the tracker, ensuring that its status is known in real time. This strategy has helped us maintain consistency within the network by promptly updating the tracker's list of active nodes.

- **Performance Optimization:**

  Challenge: File download speeds are directly influenced by network latency and the efficient distribution of file chunks.
  Solution: We optimized performance by incorporating latency

checks and parallel transfers. By evaluating the latency of individual peers, our system prioritizes connections with faster responses. Moreover, enabling parallel downloads of file chunks from multiple peers significantly reduces overall transfer times and improves responsiveness.

Challenges Still to be Addressed

- **Fault Tolerance for Tracker:**

  Challenge: The tracker is a central component that maintains peer and file location information. If the tracker fails, the entire system loses the ability to coordinate new peer discoveries or manage ongoing transfers.
  Current Status: Our current design does not provide a failover mechanism for the tracker, making it a single point of failure. Enhancing fault tolerance for the tracker—possibly through redundancy, replication, or transitioning to a fully decentralized model—remains an important future direction.

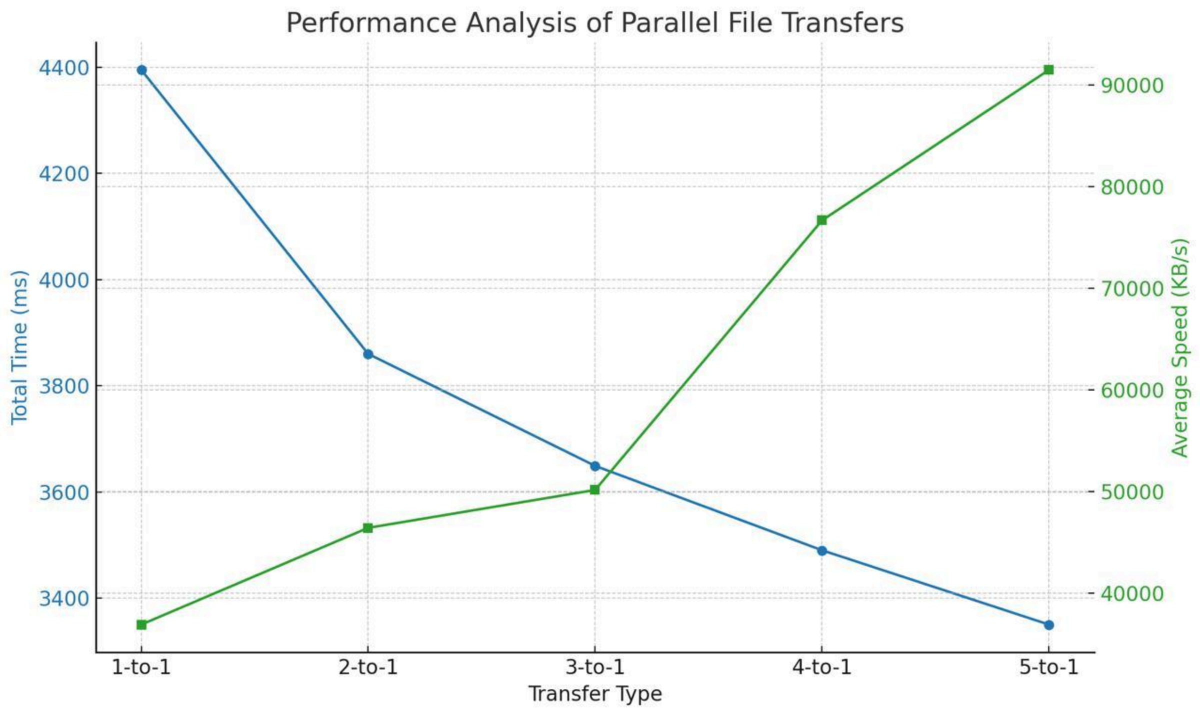
- **Scalability:**

  Challenge: As the network grows, the load on the tracker and the efficiency of peer discovery might be affected by an increasingly large number of peers and shared files.
  Current Status: Although our decentralized coordination strategy alleviates some strain by enabling direct peer-to-peer transfers, the tracker's central role for initial discovery could become a bottleneck in larger networks. Future improvements will explore additional scalability strategies, such as dynamic tracker partitioning or distributed hash tables (DHT) for decentralized discovery.

7. Results and Performance Analysis:

1. **Case : Local Multi-Peer Performance Test :**
   We tested our P2P file-sharing system by **running all peers and the tracker on a single device locally**. This setup simulates multiple uploaders (from 1-to-1 up to 5-to-1) sending file chunks to one downloader. The test was conducted using a **200 MB file** to measure performance.

Transfer Type	Total Time (ms)	Avg Speed per Node (KB/s)	Combined Speed (KB/s)	Speedup vs 1-to-1
1-to-1	4395	36933	36933	Baseline
2-to-1	3860	23225	46450	~12% faster
3-to-1	3649	16725	50175	~17% faster
4-to-1	3490	19180	76700	~21% faster
5-to-1	3350	18300	91500	~24% faster



Key Observations:

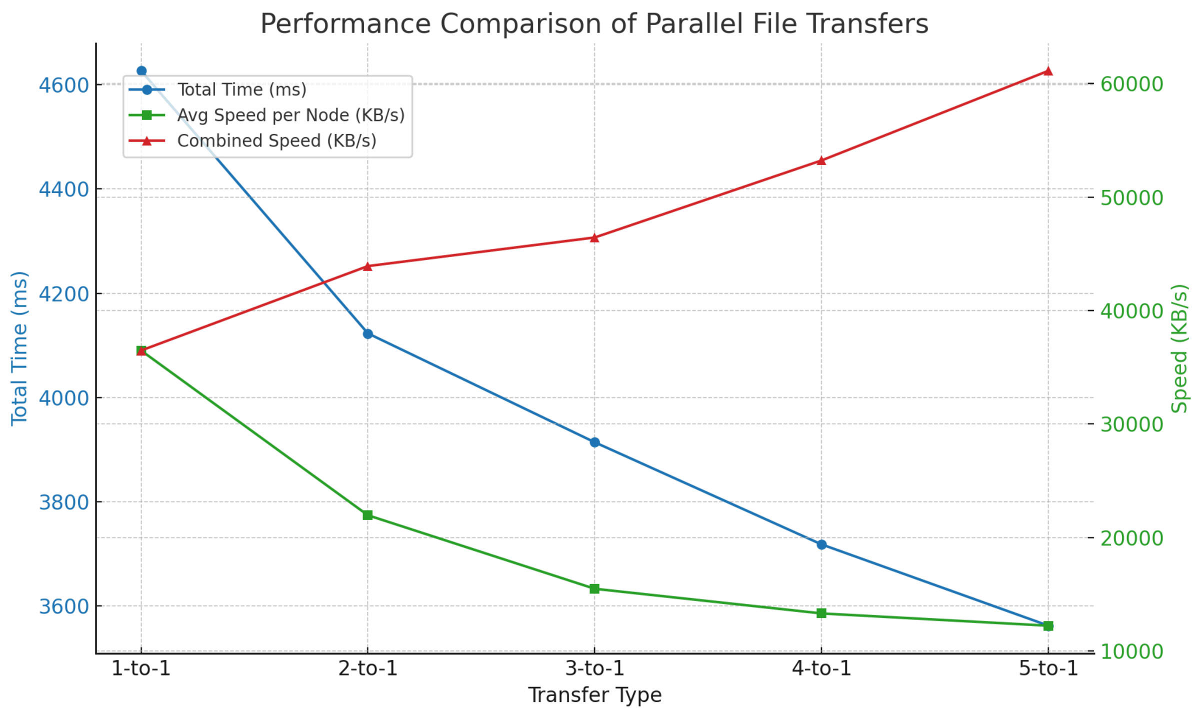
- Faster Downloads: **Transfer time improved from** 4395 ms to 3350 ms**, with up to** ~24% speedup.

- Higher Throughput: **Combined speed increased from** 36,933 KB/s to 91,500 KB/s **as peers increased.**

- Efficient Distribution: **Even though individual peer speed slightly dropped,** parallel chunk transfers **improved overall performance.**

2. **Case : Local Network-Based Multi-Peer Transfer Test :**
   In this case, we evaluated the performance of our P2P file-sharing system over a real network setup, where the tracker and **all peer nodes communicated through Local Network** across devices. The aim was to analyze performance under more realistic network conditions compared to local execution. The test was conducted using a **200 MB file** to measure performance under moderate load.

Transfer Type	Total Time (ms)	Avg Speed per Node (KB/s)	Combined Speed (KB/s)	Speedup vs 1-to-1
1-to-1	4627	36481	36481	Baseline
2-to-1	4123	21961	43922	~12.2% faster
3-to-1	3914	15481	46443	~15.4% faster
4-to-1	3718	13309	53236	~20.3% faster
5-to-1	3562	12227	61135	~24.6% faster



Key Observations:

Consistent Speedups: As the number of uploaders increased from 1 to 5, we observed speed improvements of up to ~24.6%, showing clear benefits from parallel data transfers.

Scalable Bandwidth Use: Combined speed steadily increased with more peers involved, indicating efficient utilization of available network bandwidth through simultaneous chunk sharing.

Reduced Transfer Time: Total transfer time consistently dropped as more nodes participated, demonstrating that the system scales well and maintains efficiency under increasing peer load.

Conclusion:

Running the system across networked devices confirmed that P2P file sharing remains robust and efficient outside local environments.

Speedups and improved throughput show that the system can scale well in practical, distributed settings, delivering better performance than traditional methods.

8. Future Improvements and References:

I. **Resumable Downloads:**
Enable partial file downloads by persisting chunk metadata. This would allow the system to resume interrupted downloads without needing to restart the entire file transfer process.

II. **Persistent Node State:**
Implement local storage for node information and file ownership records. This would facilitate recovery after a system restart, ensuring that nodes can quickly regain their operational status.

III. **Enhanced Security:**
Incorporate encryption for peer-to-peer communications (e.g., employing DTLS over UDP) to safeguard data in transit. Additionally, introducing node authentication measures would help prevent spoofing and unauthorized access.

IV. **Decentralized Tracker Architecture:**
Replace the centralized tracker with a Distributed Hash Table (DHT)-based solution to eliminate the single point of failure. A decentralized approach could also be paired with tracker federation and load balancing via consistent hashing.

V. **Robust Retry Mechanism:**
Add a retry logic with exponential backoff for failed chunk transfers. By limiting the number of retries and rotating through multiple peers when available, the system can improve reliability without overloading the network.