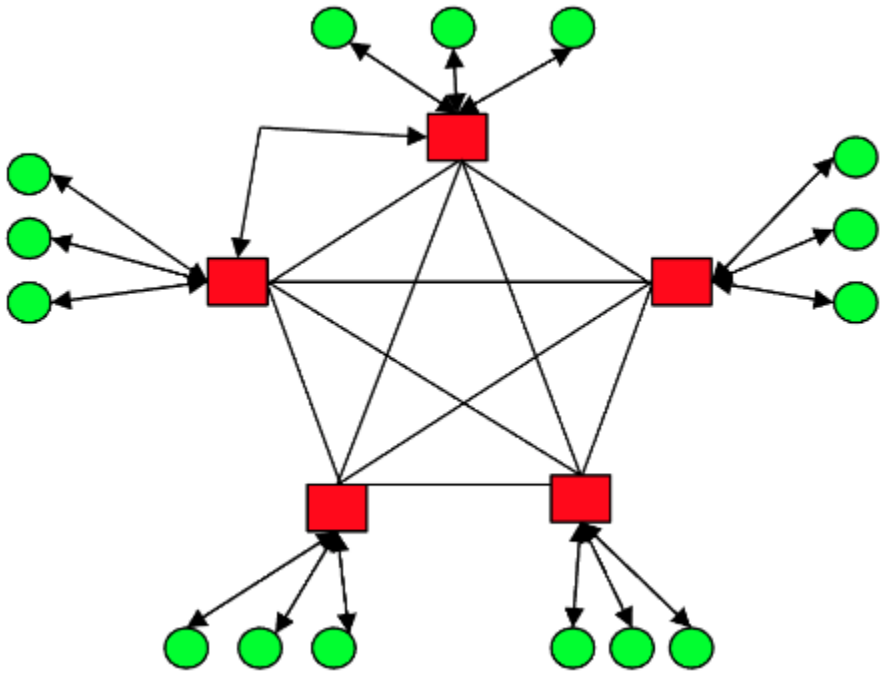
Design Document: Maintaining File Consistency in a Hierarchical Gnutella-Style P2P System Using Sockets

This document outlines the design of a file consistency system using sockets. It emphasizes scalability, modularity, and configurability, making it suitable for testing and further enhancements.



---

1. Objective
This document describes the design of a hierarchical P2P system using a Gnutella-style architecture implemented with sockets. The focus is on maintaining file consistency through push-based invalidation and pull-based polling mechanisms.

The primary objectives are:
1. Ensure consistent file state across nodes in the network.
2. Test the effectiveness of both push-based and pull-based consistency mechanisms.
3. Provide flexibility to configure and test the system using dynamic settings such as TTR (Time-to-Refresh).

---

2. System Architecture

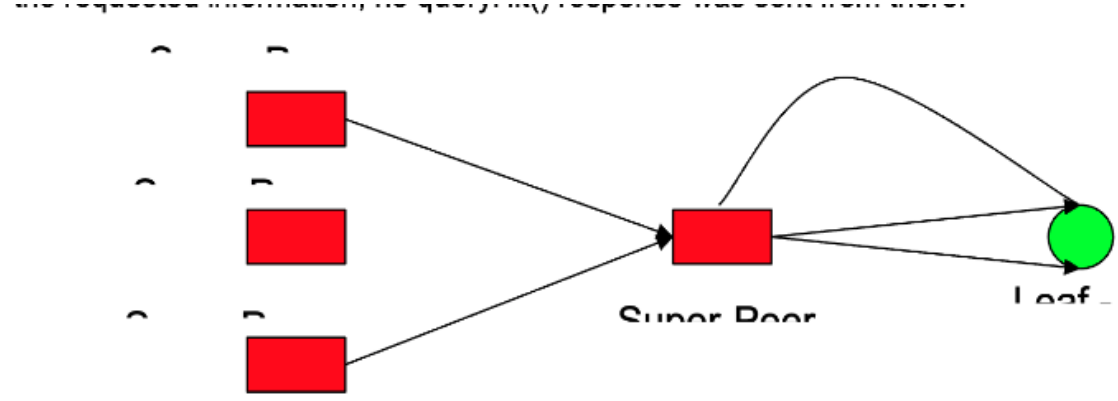
1. Super-Peers:
   - Act as intermediaries between leaf nodes and the network.
   - Maintain a registry of files from connected leaf nodes.
   - Facilitate file queries and handle broadcasts for invalidations.

2. Leaf Nodes:
   - Manage master copies and cached file versions.
   - Send queries and download files from super-peers.
   - Use push or pull mechanisms to maintain file consistency.

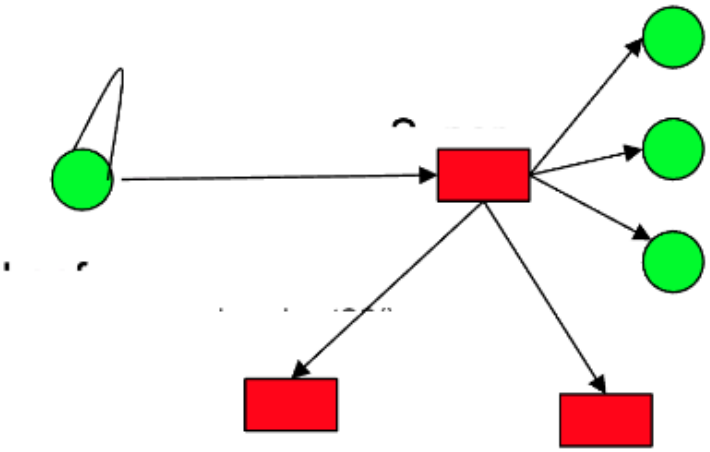
3. Network Topology:
   - Super-Peer Connections:
     - All-to-all connectivity between super-peers for redundancy.
   - Leaf Node Connections:
     - Each leaf node connects to one super-peer.

---

3. Key Features
1. File Query and Query-Hit:
   - Leaf nodes send file queries (`QUERY`).
   - Super-peers respond with query results (`QUERYHIT`) containing:
     - File name
     - Last modified time
     - Origin server details



2. Push-Based Invalidation:
   - When a file is modified, the origin node sends an `INVALIDATION` message.
   - Super-peers propagate the message to neighbors and connected leaf nodes.
   - Outdated files are marked invalid.



3. Pull-Based Polling:
   - Cached files poll the origin server periodically based on TTR.
   - Files are validated based on version numbers and timestamps.
   - Supports configurable TTR values for testing effectiveness.

4. Dynamic Configuration:
   - System settings (e.g., enabling push/pull mechanisms, TTR values) and network topology are configurable via text files:
     - system_config.txt: System-wide parameters (e.g., TTR, mechanisms).
     - network_config.txt: Super-peer and leaf-node connections.

---

4. Flow of Operations
Push-Based Invalidation:
1. A leaf node modifies a file (master copy).
2. The node broadcasts an `INVALIDATION` message with:
   - File name
   - New version
   - Origin server
3. Super-peers propagate the message to neighboring super-peers and leaf nodes.
4. Affected nodes mark outdated files as invalid.

Pull-Based Polling:
1. Cached files periodically poll the origin server when TTR expires.
2. The origin server compares the cached file's version and timestamp with the master copy.
3. Responses:
   - VALID: File is up-to-date.
   - INVALID: File is outdated and marked invalid.

---

5. Implementation
1. Data Structures:
  - FileEntry:
    - Tracks file metadata (name, version, validity, TTR).
  - SuperPeer:
    - Maintains registries of connected leaf nodes and their files.
  - LeafNode:
    - Handles master and cached files with consistency checks.

2. Message Types:
  - QUERY: Sent by leaf nodes to find files.
  - QUERYHIT: Response to queries with file metadata.
  - INVALIDATION: Broadcast to notify outdated files.
  - POLL: Sent by cached files to validate with the origin server.

3. Core Components:
  - Super-Peer:
    - Registers leaf-node files.
    - Handles queries and invalidations.
    - Propagates messages across the network.
  - Leaf Node:
    - Maintains file consistency through push and pull mechanisms.
    - Sends queries and handles downloads.

8. Enhancements

1. **Dynamic Runtime Configuration**:
    - Modify settings without restarting the program.
2. **Topology Variations**:
    - Support for linear or star network topologies.
3. **Optimized TTR Values**:
    - Dynamically adjust TTR based on network load.