Assignment #1

Parth Darshan (B22CS040), Akarsh Katiyar (B22CS006)

Link: GitHub - b22cse040/P2P_GossipProtocol

Overview

Overview of the System

The network follows a **distributed architecture**, where:

- **Seed Nodes** act as *bootstrap servers* that help new peers discover others. They keep track of active peers, distribute peer lists, and remove inactive peers.
- **Peer Nodes** register with seed nodes, establish direct peer-to-peer connections, and exchange information through *gossip-based messaging*.

The system ensures **fault tolerance, scalability, and decentralized communication** by implementing **gossip protocols, periodic peer checks, and failure detection mechanisms**.

Core Functionality and Flow

Step 1: Network Initialization

- The network starts with **one or more Seed Nodes**, which listen for peer connections and maintain a list of active nodes.
- Peer nodes are created dynamically and register themselves with one or more Seed Nodes.

Step 2: Peer Discovery & Registration

- A peer reads the **config file** to find available Seed Nodes.
- It connects to a Seed Node, registers itself, and receives a **subset of existing peers** (following a power-law distribution).
- The peer then establishes **direct connections** with some of these nodes.

Step 3: Gossip Protocol for Information Propagation

- Each peer periodically **sends gossip messages** to its connected peers.
- Gossip messages contain information about peer status, network changes, and updates.
- This ensures that **network knowledge spreads efficiently** without overloading individual nodes.

Step 4: Peer Health Monitoring & Failure Detection

- Peers regularly **ping each other** to check if they are alive.
- If a peer fails **three consecutive pings**, it is reported as *dead* to the Seed Node.
- The Seed Node removes the failed peer from its list and informs other peers.

Step 5: Continuous Network Operation

- The system runs indefinitely, handling new connections, peer departures, and information exchange dynamically.

Why This Approach?

Scalability → New peers can join seamlessly, and the network adapts dynamically.

Fault Tolerance → If some peers or Seed Nodes fail, the system continues operating without a single point of failure.

Efficiency → The **gossip protocol** reduces the need for centralized communication, ensuring fast and lightweight updates.

Decentralization → Once connected, peers communicate directly, minimizing dependence on Seed Nodes.

Real World Applications

his type of **P2P architecture** is commonly used in:

- **Blockchain networks** (e.g., Bitcoin, Ethereum) for decentralized transactions.
- **File-sharing systems** (e.g., BitTorrent) to distribute data efficiently.
- **Decentralized messaging platforms** where users communicate without central servers.
- **IoT networks** to maintain distributed communication between connected devices.

Code Description:

SeedNode.cpp : Class SeedNode

This class represents a seed node in a peer-to-peer network, responsible for maintaining a list of active peers, removing dead peers, and distributing peer lists to newly connected nodes. The class handles client connections, logs peer activities, and ensures network stability through efficient peer selection.

Private Methods

log_message(const string& message)

1. This function logs messages to the console and the peer network log file. It uses a mutex lock to ensure thread safety while logging information, such as peer connections, dead peer removals, and sent peer lists.

remove_dead_peer(const string& dead_peer)

1. Removes a peer from the peer list when it is detected as dead. It uses a mutex lock to safely erase the peer from the vector and logs the removal event.

get_power_law_peers()

1. Returns a subset of peers following an approximate power-law distribution. It randomly shuffles the peer list and selects a fraction of them. The function ensures a diverse and scalable peer network for new connections.

manage_connection(int client_socket, string client_ip)

1. Handles a client connection, receiving registration messages and adding new peers to the list. If a client reports a dead node, it is removed from the list. The function also sends a peer list to the connecting node based on a power-law selection strategy.

Public Methods

SeedNode(string ip, int p)

Constructor that initializes the seed node with an IP address and port number. It also opens a log file to record network events.

run()

Starts the seed node server by creating a socket, binding it to a port, and listening for incoming connections. For each new connection, it spawns a separate thread to handle client interactions. The function ensures the server runs indefinitely and properly cleans up resources on termination.

peer.cpp: Class PeerNode

The PeerNode class manages a peer-to-peer network by handling peer registration, gossip message propagation, and failure detection. It maintains a list of connected peers, logs messages, and interacts with seed nodes for network stabilization.

getConnectedPeers()

1. Returns the list of connected peers in a thread-safe manner using a mutex lock. This function allows access to the network topology without data inconsistencies.

PeerNode(int port)

1. Constructor that initializes the peer node with a specified port. It also writes an initialization message to a log file (peer_network.txt) to track network status.

logMessage(const string &message)

1. Logs a given message to peer_network.txt in a thread-safe manner. It also outputs the message to the console for real-time monitoring.

readConfig(const string &filename)

1. Reads the seed node addresses from a given configuration file. It stores these addresses in the seedsAddresses set for later use.

registerWithSeeds()

1. Contacts seed nodes to register itself in the network. It extracts IP addresses and ports from the config.txt file and attempts connections with a minimum of one seed node.

parseAndConnectPeers(const string &peerList)

1. Parses a list of peers received from a seed node and attempts to establish connections with them, ensuring the peer does not reconnect to itself.

sendGossipMessage()

1. Periodically sends gossip messages containing timestamps and its own address to all connected peers every 5 seconds. This helps in propagating network status updates.

pingPeers()

1. Pings connected peers every 13 seconds to check their availability. If a peer fails three consecutive pings, it is reported as a dead node and removed from the connection list.

reportDeadPeer(const string &peer)

1. Informs all seed nodes about a failed peer by sending a Dead Node message. This helps in maintaining an updated network state.

connectToPeer(string peerIp, int peerPort, string message)

1. Attempts to establish a TCP connection with a given peer and sends a message. It also includes detailed debugging statements to track the connection status.

run()

1. Initializes the peer node, creates a listening socket, and starts gossip and ping monitoring threads. It ensures the peer remains active while responding to network events.

Simulation.cpp

startSeedNode(string ip, int port, ofstream &graphFile)

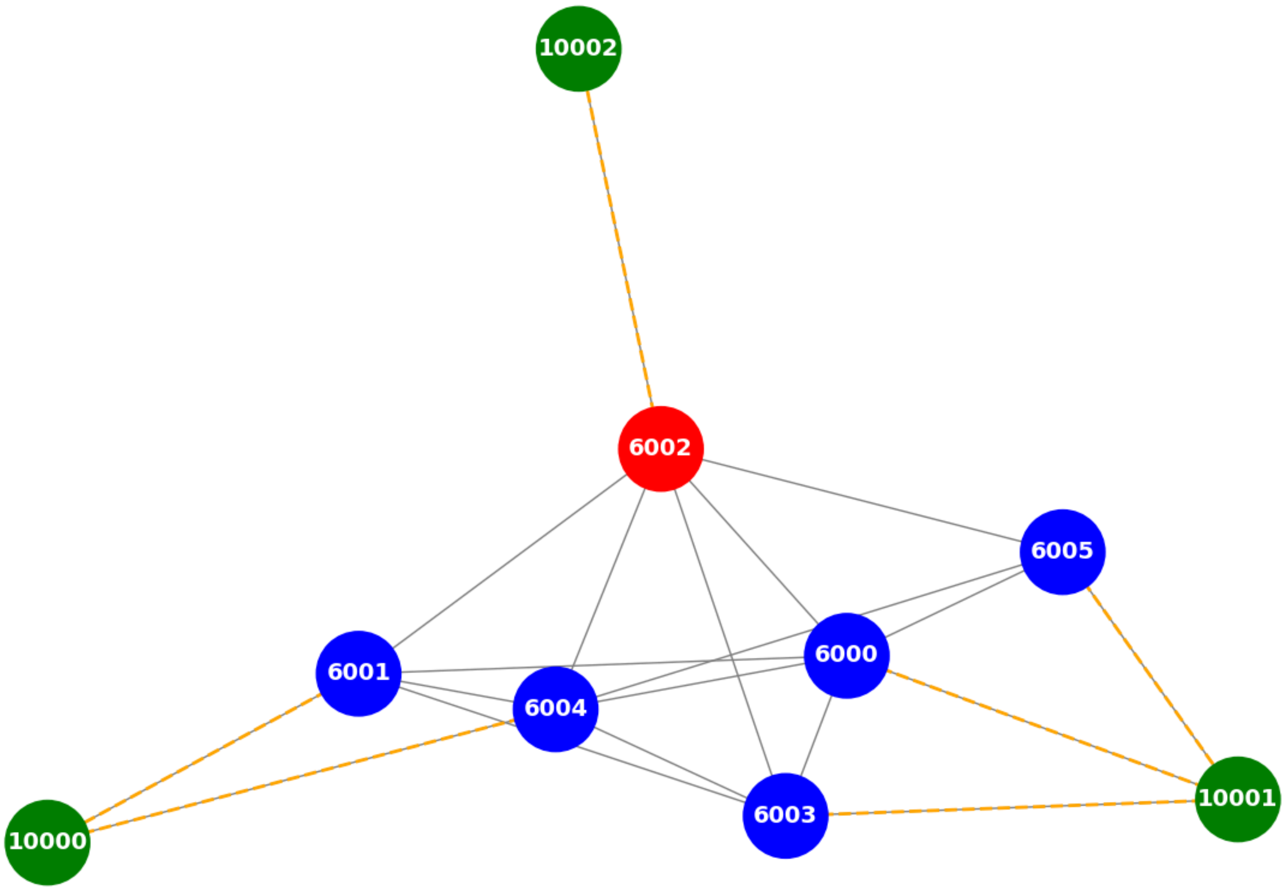
Launches a SeedNode on the given IP and port, logging its creation in the graph file while ensuring thread safety with a mutex.

startPeerNode(int port, ofstream &graphFile)

1. Creates a PeerNode, registers it with seed nodes, connects to discovered peers, starts gossip messaging in a separate thread, and runs its event loop.

main()

1. Manages the startup sequence of the network by creating multiple SeedNodes and PeerNodes, ensuring they are properly initialized, connected, and logged in the graph file before execution.


10002
6002
6005
6000
6001
6004
6003
10001
10000