Peer-to-Peer Network Simulation: Gossip Protocol and Liveness Check

Mayank Bansal(B22CS070), Yatharth Verma(B22CS064)
Instructor: Nitin Awathare

February 16, 2025

Abstract

This report details the implementation of a peer-to-peer (P2P) network simulation using a gossip protocol to broadcast messages and check the liveness of connected peers. Each peer connects to a subset of other peers in the network, and seeds maintain peer lists to bootstrap new peers. The project focuses on dynamic peer discovery, gossip message propagation, and handling peer failures through liveness checks.

1 Introduction

You can find the complete project on our GitHub repository at: https://github.com/Bansal0527/CN-Gossip-Protocol.

The objective of this project is to build a Gossip protocol over a peer-to-peer (P2P) network to broadcast messages and verify the liveness of peers. Liveness, in this context, refers to a peer being online and responsive to messages. Each peer connects to a subset of other peers in the network to create a connected graph, where communication happens through gossip messages.

Seed nodes maintain a list of peers that can be accessed by new peers to join the network. The peers periodically ping each other to check if their neighbors are alive and report any dead nodes to the seed.

2 Network Setup

The network consists of two types of nodes: Seed Nodes and Peer Nodes. Seed nodes act as a central coordinator and maintain a Peer List (PL) of connected peers, while Peer nodes handle the transmission of gossip messages and perform liveness checks.

2.1 Seed Nodes

Seed nodes are responsible for keeping track of active peers in the network. They maintain a Peer List (PL) and distribute this list to new peers when they

connect. Upon receiving a message that a peer has become unresponsive, the seed nodes update the PL and remove the dead peer.

2.2 Peer Nodes

Peer nodes connect to seed nodes to receive a list of peers in the network. They randomly choose a subset of peers, ensuring that the network degree follows a power-law distribution. Peer nodes broadcast gossip messages and periodically ping connected peers to check their liveness.

3 System Design

4 Load Handling

To manage the load on the network, each peer limits the number of active connections it maintains. This is achieved using the `limit_connection` function:

```
def limit_connection(self, complete_peer_list):
    """
    Limits the number of peer connections and starts connections with a subset of peers.
    """
    if len(complete_peer_list) > 0:
        limit = min(random.randint(1, len(complete_peer_list)), 4)
        selected_peer_nodes_index = random.sample(range(len(complete_peer_list)), limit)
        self.start_peer_connection(complete_peer_list, selected_peer_nodes_index)
```

The `limit_connection` function selects a random subset of peers, with a maximum of 4 connections. This helps balance the load and maintain a scalable peer-to-peer network.

4.1 Seed Node Design

Seed nodes create a TCP socket, bind to an IP and port, and listen for connection requests from peers. Each peer registers with at least $\lfloor (n/2) \rfloor + 1$ seed nodes, which then distribute their Peer List (PL). The seed nodes remove dead peers from the list based on reports from other peers.

4.2 Peer Node Design

Peer nodes first register with seed nodes and retrieve a list of peers from them. The peer randomly selects a subset of peers and establishes TCP connections with them. Each peer node maintains a Message List (ML) to track the messages it has sent or received, preventing duplicate message propagation.

The peer node design is multi-threaded, with threads dedicated to listening for messages, gossip message generation, and liveness checks.

4.3 Multi-threading

- **Listening Thread**: Handles incoming gossip messages from other peers.
- **Gossip Thread**: Generates gossip messages every 5 seconds and forwards them to connected peers.
- **Liveness Thread**: Sends ping messages every 13 seconds to check the liveness of connected peers.

5 Gossip Protocol

The gossip protocol ensures that each message travels through the network only once. When a peer generates a message, it propagates the message to all adjacent peers. Upon receiving a message, a peer adds the message to its Message List (ML) and forwards it to all peers except the one from which it received the message.

A peer stops generating messages after 10 messages have been sent.

6 Liveness Checking and Dead Node Reporting

Peers periodically ping each other to verify their liveness. If three consecutive pings fail, the peer marks the node as dead and informs the seed node. The seed node then removes the dead node from its Peer List.

Malicious peers may attempt to launch attacks by falsely reporting nodes as dead. A potential solution to this issue is to cross-verify the liveness of peers with multiple sources before marking them as dead.

7 Handling Functionality

Each peer in the network is responsible for managing incoming and outgoing gossip messages as well as checking the liveness of its neighbors. The handle functionality involves multiple threads that perform distinct roles:

7.1 Listening Handle

The listening handle is a thread dedicated to receiving gossip messages from connected peers. When a message is received, it is added to the Message List (ML), and if it is new, it is forwarded to other peers.

7.2 Gossip Handle

The gossip handle periodically generates new messages and forwards them to connected peers. It ensures that the same message is not sent multiple times to the same peer by checking the ML before forwarding.

7.3 Liveness Handle

The liveness handle periodically pings connected peers to check if they are still alive. If a peer does not respond to three consecutive ping messages, it is marked as dead, and a report is sent to the seed node.

8 Output and Results

Each seed node logs connection requests, peer list updates, and dead node reports. Peer nodes log gossip messages, liveness checks, and dead node reports. Logs are displayed on the console and stored in output files.

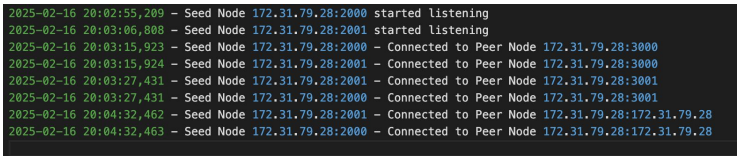

Figure 1: Screenshot of Seed Node Execution

Figure 2: Screenshot of Peer Node Execution

Figure 3: Screenshot of Dead Node detection

9 Conclusion

This project demonstrated the implementation of a P2P network using a gossip protocol for message propagation and liveness checks. By simulating dynamic peer discovery and handling peer failures, the network maintains robust connectivity and fault tolerance.