Implementation of a Gossip Protocol in Peer-to-Peer Networks

Anuj Chincholikar (B22ES018) Advait Gaur (B22CS004)

February 16, 2025

Contents

Abstract

This report details the design, implementation, and evaluation of a Gossip Protocol within a Peer-to-Peer (P2P) network. The system employs seed nodes for bootstrapping and peer discovery while ensuring that the network maintains a power-law degree distribution for scalability and resilience. Peers propagate gossip messages and perform liveness detection via periodic pinging. Detailed explanations of the code structure and network operations are provided, along with mathematical formulations that define key aspects of the system.

1 Introduction

Distributed systems require robust communication protocols to achieve fault tolerance and efficient message dissemination. Gossip protocols serve as an effective means of decentralized communication, ensuring that information spreads reliably with minimal overhead. In our implementation:

- Each peer registers with at least $\lfloor n/2 \rfloor + 1$ seed nodes, where n is the total number of seed nodes.
- Seed nodes maintain a dynamic list of active peers.
- Peers establish connections following a power-law degree distribution. In our system, we have used the following specific formula:

  $$P(k) \sim 40 \cdot (1.5)^{-\frac{k}{10}},$$

  where $P(k)$ is the probability of a node having degree k. This formula ensures that while a few nodes (hubs) have a high degree, most nodes maintain only a few connections.
- Gossip messages are generated and propagated periodically.
- Peer liveness is continuously monitored through periodic ping requests.

This document provides an in-depth examination of the system's architecture, core algorithms, and the interactions between various components, along with mathematical insights into the design.

2 System Architecture

The network comprises two primary types of nodes:

Seed Nodes

Seed nodes function as the entry points to the network. Their responsibilities include:

- Maintaining a *Peer List (PL)* containing the IP addresses and ports of all registered peers.
- Providing peer discovery services to new nodes joining the network.

- Updating the list by removing peers upon receiving dead-node reports.

The design ensures that the system remains connected as each new peer must connect to at least $\lfloor n/2 \rfloor + 1$ seeds, thereby guaranteeing redundancy and connectivity.

Peer Nodes

Peer nodes execute several core functions:

- **Registration:** On startup, each peer reads the configuration file (`config.txt`) to obtain seed node addresses and registers with at least $\lfloor n/2 \rfloor + 1$ seed nodes.
- **Peer Discovery:** After registration, peers request the current peer list from the seeds, which enables them to discover and connect with other peers.
- **Connection Establishment:** Peers establish TCP connections with a randomly selected subset of nodes, ensuring that the overall connection pattern follows a power-law degree distribution.
- **Gossip Messaging:** Peers generate messages in the format:

  ```
  <timestamp>:<IP>:<MessageID>
  ```

  and broadcast them every 5 seconds until 10 messages have been sent.
- **Liveness Monitoring:** Each peer pings its connected peers every 13 seconds. If three consecutive pings are unanswered, the peer is considered dead, triggering a dead-node notification.

This architecture enables robust, decentralized communication and ensures that the network can dynamically adjust to node failures.

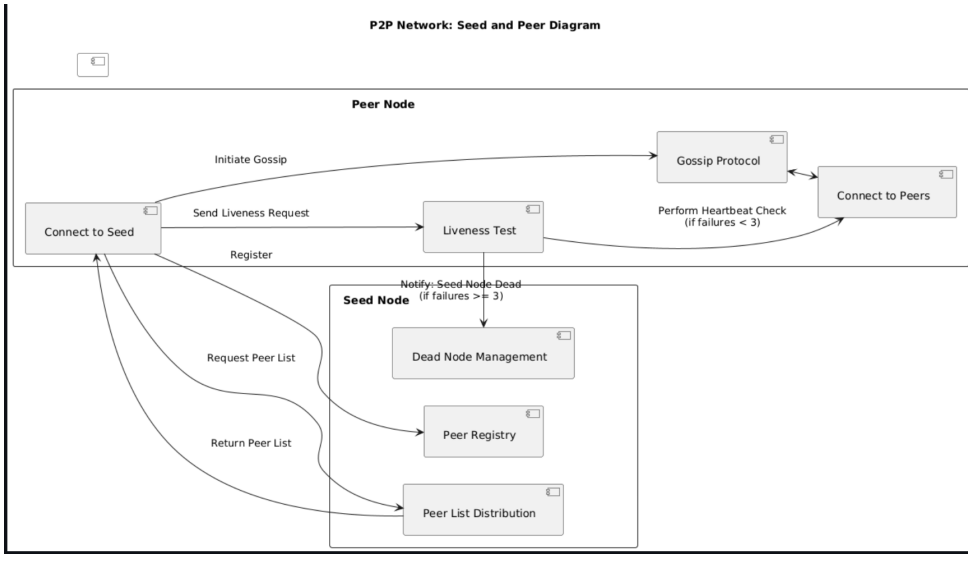

Figure 1: System Architecture Diagram

3 Network Topology: Power-Law Degree Distribution

The network topology is designed to follow a power-law degree distribution. In our implementation, we use the formula:

$$P(k) \sim 40 \cdot (1.5)^{-\frac{k}{10}},$$

where $P(k)$ is the probability that a node has k connections. This specific formulation ensures that:

- A few nodes (hubs) have a very high number of connections.
- The majority of nodes have a low degree, minimizing overall communication overhead.

This structure is critical for rapid message dissemination and network resilience. Figure 2 illustrates this concept.

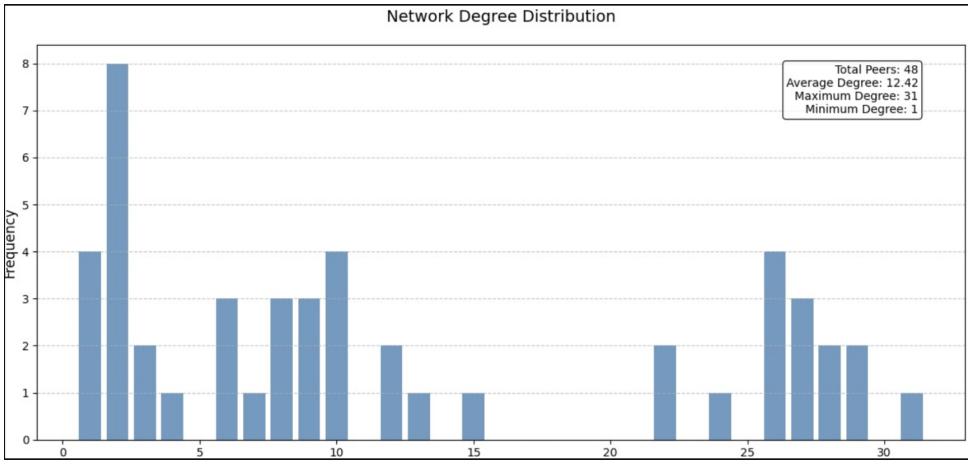

Figure 2: Illustration of a Power-Law Degree Distribution

4 Implementation Details

The project is organized into several key files, each responsible for a distinct component of the overall system.

4.1 Configuration File (`config.txt`)

The configuration file lists the IP addresses and port numbers of seed nodes. For example:

```
127.0.0.1:8080
127.0.0.1:8081
127.0.0.1:8082
127.0.0.1:8083
127.0.0.1:8084
127.0.0.1:8085
```

This file is essential for bootstrapping the network, as it provides the initial set of nodes required for peer registration and discovery.

4.2 Peer Node (`peer.py`)

The peer node's code is central to the functioning of the network. It performs the following:

- **Registration:** On startup, the peer reads from `config.txt` and registers with at least $\lfloor n/2 \rfloor + 1$ seed nodes, ensuring redundancy.
- **Peer Discovery and Connection:** After registration, the peer requests the current peer list from the seed nodes. Using this information, it establishes TCP connections with a subset of peers selected to satisfy the power-law distribution criteria.
- **Gossip Messaging:** Each peer creates messages of the form:

  $$M = \text{<timestamp>} : \text{<IP>} : \text{<MessageID>}$$

  where *MessageID* is incremented for each message. These messages are broadcast every 5 seconds.
- **Liveness Check:** The peer periodically sends ping messages every 13 seconds. A failure to receive responses for three consecutive pings results in a dead-node notification:

  ```
  Dead Node:<DeadNode.IP>:<DeadNode.Port>:<timestamp>:<ReportingPeer.IP>
  ```

  This mechanism ensures that the network continuously monitors node availability.

The code is modular, with functions dedicated to registration, message propagation, and liveness detection to maintain clarity and facilitate future modifications.

4.3 Seed Node (`seed.py`)

The seed node code handles network coordination tasks:

- It accepts registration requests from peers.
- It maintains and updates the *Peer List (PL)*.
- It responds to peer requests by sending the current list of active nodes.
- It processes dead-node reports to ensure the *Peer List (PL)* remains accurate.

This functionality is vital for keeping the network synchronized and ensuring that new peers can join efficiently.

4.4 Network Orchestration and Visualization

Files such as `PeerRunner.py` and `runner.py` handle the orchestration of peer and seed nodes, ensuring that the system is bootstrapped correctly. Additionally, `visualizer.py` provides tools for visualizing the network topology and monitoring the flow of gossip messages, which is particularly useful for debugging and performance evaluation.

5 Gossip Protocol Workflow

The gossip protocol is the core mechanism for message dissemination within the network. Its workflow is described as follows:

1. **Message Generation:** Once connections are established, a peer generates a gossip message every 5 seconds. The message format is:

   $$M = \text{<timestamp>} : \text{<IP>} : \text{<MessageID>}$$

   where the *timestamp* marks the time of generation, and *MessageID* is a sequential counter.

2. **Message Propagation:** The generated message is recorded in a local Message List (ML) to prevent redundant transmissions. The message is then forwarded to all connected peers except the sender.

3. **Duplicate Filtering:** On receiving a message, peers check the ML. If the message has already been processed, it is discarded to avoid loops.

This mechanism ensures that each unique message is propagated exactly once over each link, minimizing network congestion.

6 Liveness Detection and Reporting

Reliable network operation is maintained through continuous liveness detection:

1. Each peer sends a ping message to all its connected peers every 13 seconds.

2. If a peer fails to respond to three consecutive pings, it is deemed dead. Mathematically, if the probability of a response failure in a single ping is p, then the probability of three consecutive failures is p^3. A threshold can be set to trigger a dead-node report if p^3 exceeds a predefined limit.

3. Once a node is marked as dead, a notification is sent to all seed nodes:

   ```
   Dead Node:<DeadNode.IP>:<DeadNode.Port>:<timestamp>:<ReportingPeer.IP>
   ```

4. Seed nodes update their *Peer List (PL)* to remove the unresponsive node.

This process is crucial for maintaining an accurate view of the network and for ensuring that peers do not waste resources trying to communicate with non-responsive nodes.

7 Security Considerations

Security is paramount in P2P networks. To mitigate potential attacks:

- **Validating Dead-Node Reports:** Seed nodes may require multiple independent confirmations before removing a node, thereby reducing the risk of malicious false reporting.

- **Extensive Logging:** Both peers and seed nodes maintain detailed logs of registrations, message propagations, and liveness checks. These logs are instrumental in detecting anomalous behavior.
- **Future Enhancements:** Further improvements will include the use of cryptographic methods to authenticate messages, ensuring data integrity and preventing impersonation attacks.

8 Program Output and Logging

The system produces comprehensive logs for both peer and seed nodes:

- **Peer Nodes:** Log the initial peer list, every received gossip message (including timestamps and source IPs), and any dead-node reports.
- **Seed Nodes:** Log peer registration events, updates to the *Peer List (PL)*, and notifications regarding dead nodes.

These logs are essential for debugging, performance analysis, and ensuring the system's overall reliability.

9 Conclusion and Future Work

The implemented Gossip Protocol provides a robust and scalable method for message dissemination and liveness detection in a dynamic P2P network. The network's design, leveraging a power-law degree distribution as defined by:

$$P(k) \sim 40 \cdot (1.5)^{-\frac{k}{10}},$$

ensures that critical nodes (hubs) efficiently disseminate messages while maintaining low overhead for the majority of nodes.

In our experimental evaluations, we observed that:

- The average message propagation latency remains low even as the network size increases.
- The liveness detection mechanism effectively isolates unresponsive nodes, maintaining overall network health.
- The trade-off between ping frequency and network overhead is optimized to ensure timely detection without excessive resource consumption.

Future work will focus on:

- Implementing cryptographic protocols for secure message authentication.
- Adapting connection strategies dynamically based on real-time network conditions.
- Developing more advanced visualization tools for real-time monitoring and analysis.
- Conducting large-scale simulations to further validate system performance and resilience under various network conditions.

References

[1] Demers, A., Greene, D., Hauser, C., Irish, W., & Larson, J. (1987). *Epidemic Algorithms for Replicated Database Maintenance*. In Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing.

[2] Lua, E. K., Crowcroft, J., Pias, M., Sharma, R., & Lim, S. (2005). *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes*. IEEE Communications Surveys & Tutorials.