Simplified Gossip Protocol

Project Report

Student Names: Arina Zimina, Karina Siniatullina,

Adelina Karavaeva, Egor Agapov

Date: 30.04.2025

1 Introduction

The **Gossip protocol** is a protocol that allows designing highly efficient, secure and low latency distributed communication systems (P2P). The inspiration for its design has been taken from studies on epidemic expansion and algorithms resulting from it.

The gossip protocol is very important in distributed systems because it helps **nodes** (computers, servers, or processes) **share information quickly**, **reliably**, and **without a central coordinator**. Here's why it's critical:

- **Scalability:** Gossip scales really well — even with thousands of nodes — because each node only talks to a few others at a time.
- **Fault tolerance:** Nodes can fail or go offline, but gossip ensures the system can still spread information without depending on any single point.
- **Eventually consistent:** Perfect synchronization is hard in distributed systems, so gossip allows nodes to eventually reach the same state without requiring immediate consistency.
- **Low overhead:** The communication is lightweight and randomized, so it doesn't overload the network.

A few important use cases for gossip protocols:

1. **Membership tracking:** Nodes use gossip to find out which other nodes are alive, dead, or new in the system (example: Amazon DynamoDB).
2. **State dissemination:** Systems like Apache Cassandra use gossip to spread metadata (like schema changes, load info) across all nodes.
3. **Failure detection:** If a node crashes, gossip helps quickly alert the rest of the system so they can reroute traffic or rebalance data.
4. **Blockchain and cryptocurrency networks:** In Bitcoin, Ethereum, and other decentralized networks, gossip spreads new transactions and blocks across peers.

2 Methods

Our implementation of the Gossip Protocol consists of a modern web application with a clear separation between backend and frontend components. The system architecture is designed to be scalable, maintainable, and provides real-time visualization of the gossip protocol simulation.

2.1 Architecture

The project follows a client-server architecture:

- **Backend:** Implemented in Python using FastAPI framework, providing RESTful API endpoints for simulation control and data retrieval
- **Frontend:** Built with modern web technologies, offering an interactive visualization of the gossip protocol
- **Communication:** REST API with CORS support for seamless frontend-backend interaction

2.2 Simulation Implementation

The core simulation logic is implemented in the following components:

- **Node Class:** Represents individual nodes in the network with properties for:
  - Node identification
  - Data storage
  - Active/inactive states
  - Alive/dead states
- **Gossip Algorithm:** Implements the core protocol with features:
  - Random peer selection
  - Concurrent message exchange using threading
  - Fault tolerance simulation (5% node failure probability)
  - Node recovery simulation (2% recovery probability)
  - Convergence detection

2.3 Convergence and Metrics

The system tracks several important metrics:

- **Convergence:** Detected when all alive nodes have the same data
- **Round History:** Each round's state is saved, including:
  - Node states (active/inactive, alive/dead)
  - Message exchanges
  - Convergence status
- **Simulation Logs:** Detailed JSON logs for each simulation round

2.4 Tools and Technologies

- **Backend:** Python, FastAPI, threading
- **Frontend:** Modern web stack (HTML, CSS, JavaScript)
- **Development:** Git for version control
- **Testing:** Dedicated test suite for protocol verification

3 Results

4 Discussion

5 References

1. What is the Gossip Protocol: `https://academy.bit2me.com/en/what-is-gossip-protocol/`