Gossip Scope: A Visual aid for learners

Sriram Nuthi
nuthi@usc.edu

Sashank Agarwal
sashanka@usc.edu

Abstract

The project aims to develop a visualization tool for the Gossip Distributed Protocol [4], similar to the Raftscope tool [6] used for visualizing the Raft consensus protocol. The Gossip protocol [1] is widely used in distributed systems to propagate information across a network of nodes in a decentralized manner. It is known for its scalability, fault tolerance, and eventual consistency, making it ideal for large-scale distributed environments such as peer-to-peer networks, distributed databases, and cluster management systems. Notably, major companies like Amazon Web Services (AWS) use the Gossip protocol in DynamoDB for managing membership and ensuring eventual consistency across distributed nodes. HashiCorp Consul also leverages a gossip-based protocol for cluster management and failure detection, while Apache Cassandra uses it for node discovery and failure detection in its highly scalable NoSQL database. These implementations highlight the protocol's critical role in enabling robust and scalable distributed systems across various industries.

1 Motivation

Understanding the behavior of distributed protocols like Gossip can be challenging due to their decentralized nature and probabilistic communication patterns [4]. While tools like Raftscope [6] provide an intuitive way to visualize consensus protocols like Raft, there is a lack of similar tools for the Gossip protocol. This project seeks to fill that gap by providing a dynamic visualization of how information spreads across nodes in a network using the Gossip protocol. The tool will help researchers, students, and developers gain insights into the protocol's behavior, including message propagation, node failures, and convergence.

2 Background

The Gossip protocol operates by having nodes periodically exchange information with randomly selected peers. Over time, this leads to eventual consistency across all nodes in the network. The protocol has several key features:

- Decentralization: No central authority; each node independently gossips with others.
- Probabilistic Selection: Nodes randomly select peers to share information.
- Fault Tolerance: The protocol is resilient to node failures and network partitions.
- Scalability: Suitable for large networks due to its low communication overhead.

Similarly, gossip visualization tools like WebGC[2] and GoMoChe[5] have been developed but are either limited in scope or difficult to use. This project will build on these efforts by offering an intuitive and interactive visualization tool specifically designed for the Gossip protocol.

3 Objectives

The main objectives of this project are:

1. Visualize Gossip Protocol Dynamics: Provide a real-time graphical representation of how nodes exchange information and how data propagates through the network.
2. Simulate Node Failures: Allow users to simulate node failures and observe how the protocol handles fault tolerance.
3. Track Convergence: Show how nodes eventually converge to a consistent state after multiple rounds of gossiping.

4. User Interactivity: Enable users to interact with the simulation by adding/removing nodes or changing parameters (e.g., number of neighbors each node gossips with, failure rate) to see how these factors affect the spread.

4 Design

4.1 Architecture

The *Gossip Scope* visualization tool leverages modern web technologies to provide an interactive and real-time visualization experience. It is implemented using **React.js** for the user interface and **D3.js** for dynamic graph visualization. The system comprises several key components as shown in Figure 1:

- **Graph Visualization**: Nodes are represented as circles with color-coding to indicate their state (e.g., uninformed, informed, or failed).[3]
- **State Management**: Tracks the status of each node, message propagation, and the overall network topology dynamically.
- **Interactive Controls**: Users can modify simulation parameters and trigger protocol-related events.
- **Animation System**: Visualizes the propagation of messages between nodes to illustrate how information spreads.[3]

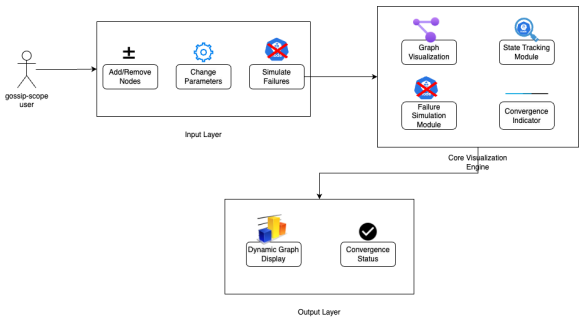

Figure 1: Architecture design of Gossip Scope

4.2 Core Features

The tool includes several interactive features designed to enhance user engagement and understanding of the Gossip protocol:

- **Node Management**: Users can add new nodes, remove existing ones, and toggle node failures to simulate faults in the network.
- **Parameter Control**: Adjustable settings for parameters such as fanout (number of nodes each node gossips with) and cycle delay (time interval between gossip rounds).
- **Visualization Modes**: Supports both automatic gossip propagation and manual message sending to observe different scenarios.
- **Path Visualization**: Displays potential communication paths between nodes, highlighting the routes taken by messages during propagation.

4.3 Modes of Operation

The Gossip Scope tool offers two distinct modes of operation to enhance usability and cater to different learning styles:

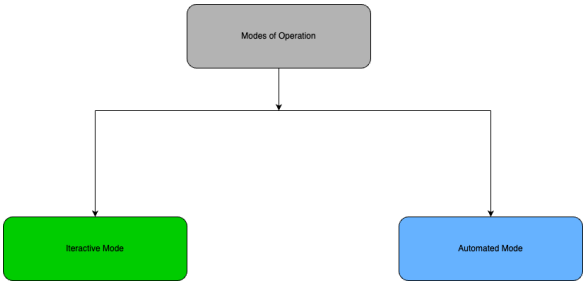

Figure 2: Modes of operation

4.3.1 Automated Mode

- This mode allows users to observe the protocol in action without requiring manual intervention.
- It can be initiated by simply clicking the **Start Gossip** button.
- The tool takes control and executes the gossip protocol automatically, enabling users to focus on understanding the dynamics of the process.

4.3.2 Interactive Mode

- This mode provides users with a hands-on, step-by-step exploration of the gossip protocol.
- Users have access to the following actions:
  - **Reset Nodes**: Resets all nodes to their default state, clearing any prior information or faults.

- **Show Paths**: Visualizes the connections between nodes, helping users understand the network topology.
- **Random Selection**: Adjusts the number of target nodes (fanout) for message propagation. Each click generates a new random selection of nodes.
- **Send Message**: Activates after a random selection is made, allowing users to propagate messages manually.

- This interactive approach empowers users to experiment with different scenarios, enhancing their understanding of the protocol's behavior and mechanics.

5 Implementation

5.1 Node Representation

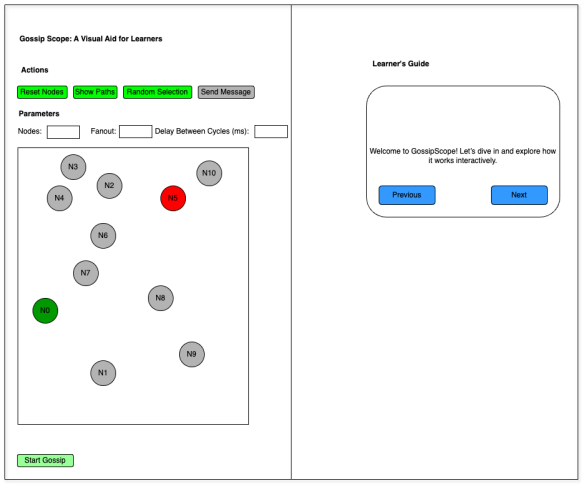

Figure 3: Gossip Scope UI Frame

The UI frame is divided into 2 components, the left section presents the interactive play ground that users can use to get there hands dirty. The right section is a learners guide that walks the learners through the concepts and necessary actions of the playground.

Each node in the network is represented as a JavaScript object with the following structure:

```
const nodes = {
  id: number,
  x: number,
  y: number,
  informed: boolean,
  failed: boolean,
  data: array
};
```

Here:

- `id`: A unique identifier for the node.
- `x, y`: Coordinates representing the node's position in the graph.
- `informed`: Indicates if the node has received the information.
- `failed`: Denotes whether the node is currently inactive.
- `data`: Stores any additional node-specific information.

5.2 Protocol Implementation

The Gossip protocol logic is implemented in JavaScript, focusing on message propagation and handling node interactions. The main function is shown below:

```
const runGossipRound = () => {
  // Select informed nodes
  const informedNodes = nodes.filter(
      node => node.informed && !node.
      failed);

  // For each informed node, select
      random targets based on fanout
  informedNodes.forEach(node => {
    const targets = selectRandomTargets(
        fanout);
    propagateInformation(node, targets);
  });
};
```

This implementation includes:

- `runGossipRound`: Executes a single round of the Gossip protocol.
- `selectRandomTargets`: Randomly selects peers based on the fanout.
- `propagateInformation`: Handles message transfer to the selected targets.

5.3 Visualization Logic

The animation system utilizes SVG to visualize message propagation. The following snippet demonstrates the animation logic:

```
<animate
  attributeName="x"
  from={source.x}
  to={target.x}
  dur={GOSSIP_SPEED}
  repeatCount="1"
/>
```

Explanation:

- `attributeName="x"`: Specifies the property being animated (e.g., horizontal position).
- `from` and `to`: Define the start and end positions for the animation.
- `dur`: Duration of the animation, controlled by `GOSSIP_SPEED`.
- `repeatCount="1"`: Ensures the animation runs once per propagation.

This design provides a dynamic visualization of how messages are exchanged across the network.

6 Results

6.1 System Implementation

The implemented visualization tool successfully demonstrates the key aspects of the Gossip protocol through an interactive web interface. The main achievements include:

1. **Interactive Visualization:** The system effectively renders network nodes with distinct color-coding:
   - **Gray:** Represents uninformed state.
   - **Green:** Indicates informed state.
   - **Red:** Denotes failed state.

6.2 Parameters Analysis

The Parameters of the tool can be adjusted accordingly to help out learners try out and understand different scenarios, summarized in Table 1.

Parameter	Impact
Node Count	Successfully handles up to 10 nodes with smooth visualization.
Fanout	Configurable from 1 to n, directly impacts convergence speed.
Cycle Delay	Adjustable from 1000ms upward for clear visualization.

Table 1: Performance Analysis of the Gossip Scope Tool

6.3 Key Observations

6.3.1 Protocol Behavior

- The visualization clearly demonstrates the logarithmic spread of information.
- Maximum rounds required follow the formula: $\log_{\text{fanout}}(\text{nodes})$.

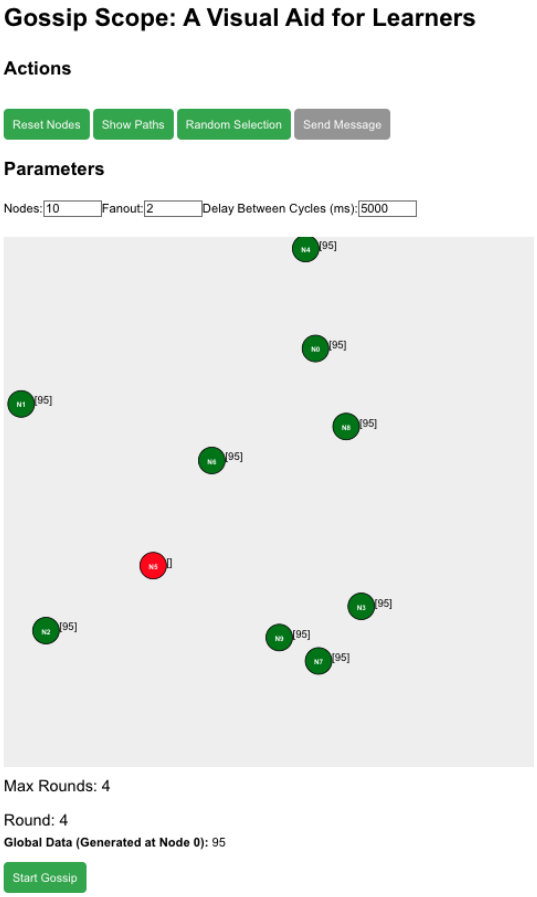

Figure 4: Result after running the Gossip protocol: Node N5 (marked in red) is initially faulty and requires four rounds to receive information from the other nodes in the cluster.

- Node failures are handled gracefully without disrupting overall propagation. As shown in Figure 4, the result after running the Gossip protocol algorithm is depicted. In this scenario, one node, N5, is initially faulty and marked in red. It takes a total of four rounds for the other nodes in the cluster to propagate the necessary information to N5, enabling it to become informed. Figure 5 illustrates the final state after the faulty node N5 recovers and rejoins the network. Once N5 is operational again, the remaining nodes in the cluster send their updated values to N5 to ensure that it is synchronized with the rest of the system. As a result, one additional round of gossiping is required to update N5, thus completing the process of information convergence across all nodes in the cluster.

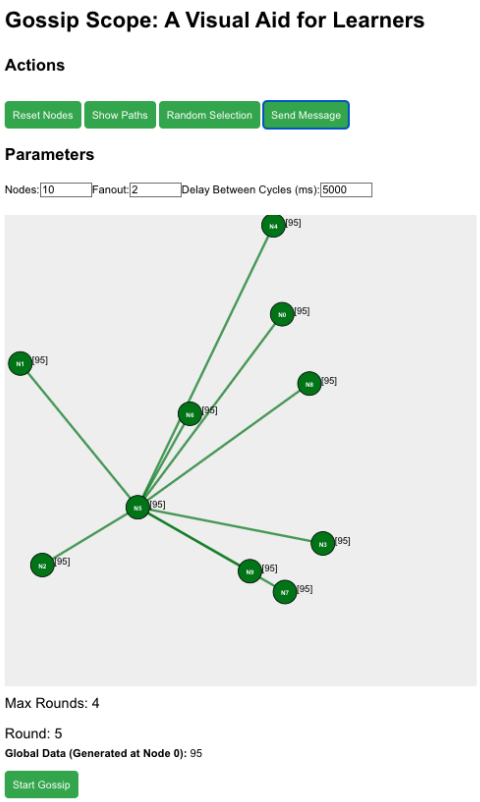

Figure 5: Convergence of all nodes after the faulty node N5 recovers: Once N5 is operational, the other nodes update it in one additional round, completing the synchronization across the network.

6.3.2 User Interaction

- Real-time parameter adjustment provides immediate feedback.
- Node status toggling effectively demonstrates fault tolerance.
- Path visualization helps understand network topology.

6.3.3 Educational Value

- Interactive guide system helps users understand protocol concepts.
- Step-by-step visualization aids in comprehending message propagation.
- Parameter experimentation enables practical learning.

The results demonstrate that Gossip Scope successfully achieves its primary goal of providing an intuitive visualization tool for understanding gossip protocols in distributed systems.

7 Future Work

To further enhance the capabilities of the Gossip Scope tool, the following features are planned for future development:

- Advanced failure scenario simulation to model complex node failure patterns.
- Network partition visualization to illustrate how the protocol handles split-brain scenarios.
- Integration with real distributed system metrics to provide practical insights into protocol performance.
- Support for different gossip protocol variants, enabling comparative analysis of their behavior and efficiency.

8 Conclusion

Gossip Scope successfully provides an interactive platform for understanding gossip protocols in distributed systems. The tool's ability to visualize message propagation, node failures, and convergence patterns makes it valuable for both educational and development purposes. By offering real-time interactivity and a user-friendly interface, the tool bridges the gap between theoretical understanding and practical application of distributed systems.

References

[1] C. Hauser W. Irish A. J. Demers, D. H. Greene and J. Larson. Epidemic algorithms for replicated database maintenance. *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, pages 1–12, 1987.

[2] Raziel Carvajal. WebGC: A Web-Based Graphical Calculator. https://github.com/raziel-carvajal/WebGC, 2024. Accessed: Oct. 31, 2024.

[3] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, pages 1–12. ACM, 1987.

[4] Anne-Marie Kermarrec and Maarten van Steen. Gossiping in distributed systems. *INRIA and Vrije Universiteit Amsterdam*, 2024. Available: https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/30673933/5661ecd4-c500-4d39-93bc-27de99f9c81f/Gossiping.pdf.

[5] Mayank Kumar, Goutham Chowdhury, et al. LAMASSR: Language Model Assisted Source Code Summarization with Rich Metadata. In *GoMoChe: Workshop on Generative Models for Code in High Energy Physics*, 2022.

[6] Diego Ongaro. RaftScope. https://raft.github.io/raftscope/index.html, 2024. Accessed: Oct. 31, 2024.