

# Evaluating Solutions to the Gossip Problem

Ketan Tomar

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfilment of the requirements  
for the degree of

MASTER OF SCIENCE

September 2023

# Abstract

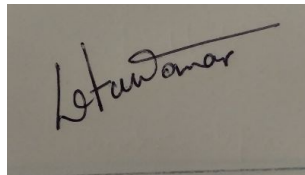
In network communications, the gossip problem serves as a pivotal model for understanding peer-to-peer information sharing. This project navigates this problem by developing and examining a range of strategies to find a near-optimal solution. The primary objective here is to enable a set of agents to disseminate their exclusive secrets through phone conversations, aiming for the fullest spread of secrets in the least amount of calls. To this end, we established a Python simulation environment where agents are defined by specific attributes, including unique secrets and a record of their communications. The agents operate under one of four protocols: "ANY," allowing free communication between all agents; "CO," in which agents can only have a one-off communication with others; "SPI," where only those holding a token can start a conversation; and "LNS," where communication happens only if there is at least one secret to be learned. The 'Gossip Simulator' class oversees the execution of these protocols, running through a predetermined number of rounds until all secrets are known to all agents or the round limit is reached. The metrics highlighted in our simulation results, such as the average number of contacts each agent had and the total number of rounds needed for the full sharing of secrets, offer a distinct performance outline for each protocol. This gives an in-depth understanding of their efficiency and prospective applications.

# Student Declaration

I confirm that I have read and understood the University's Academic Integrity Policy. I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study.

I confirm that I have not copied material from another source nor committed plagiarism nor fabricated data when completing the attached piece of work. I confirm that I have not previously presented the work or part thereof for assessment for another University of Liverpool module. I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

I confirm that I have not incorporated into this assignment material that has been submitted by me or any other person in support of a successful application for a degree of this or any other university or degree-awarding body.

A photograph of a handwritten signature in black ink on a light-colored surface. The signature appears to be 'Letawana' written in a cursive style.

SIGNATURE \_\_\_\_\_

DATE            September 22, 2023

# Acknowledgement

The author wishes to acknowledge the following people for their contribution towards the completion of this work:

- Project supervisors, Louwe Kuijer and Qiyi Tang, for their efforts and time invested in guiding me through each stage of the project.
- My parents and sister for their unwavering support.
- My friends and colleagues at the University of Liverpool.

# CODE OF CONDUCT

In this dissertation, I have diligently addressed professional issues integral to the field of Computer Science. Following the British Computer Society's (BCS) guiding principles, I have upheld the highest standards of ethical behaviour while demonstrating professional competence and skill throughout the study process. Data utilisation has been methodically ethically integrated, assuring privacy, informed permission, and ethical data processing. Human volunteers were treated with the utmost respect, and strict safeguards were put in place to protect their rights and welfare. This research has benefited my professional growth and is in line with the moral principles that support ethical behaviour in the larger professional community.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Scope . . . . .	1
1.2	Problem Statement . . . . .	1
1.3	Approach . . . . .	1
1.4	Outcome . . . . .	2
<b>2</b>	<b>Aims and Objectives</b>	<b>3</b>
2.1	Aims . . . . .	3
2.2	Objectives . . . . .	3
2.3	Objectives Achieved . . . . .	4
<b>3</b>	<b>Background</b>	<b>5</b>
3.1	Gossip Protocols . . . . .	5
3.1.1	<b>Gossip Time Protocol (GTP)</b> . . . . .	5
3.1.2	GossipFF (Fixed Fanout Gossip) . . . . .	6
3.1.3	GossipPE (Probabilistic Edge Gossip) . . . . .	6
3.1.4	GossipPB (Probabilistic Broadcast Gossip) . . . . .	7
3.2	Gossip Protocols Analysis Approach . . . . .	8
3.3	Keywords . . . . .	8
3.4	OMNET++ Simulation . . . . .	10
3.4.1	Parameters Used for Simulation . . . . .	10
3.4.2	Topologies used for OMNET++ Simulation . . . . .	10
3.5	Implemented Protocols . . . . .	11
<b>4</b>	<b>Design</b>	<b>14</b>
4.1	Data Structures and Algorithms . . . . .	14
4.1.1	Sets . . . . .	14
4.1.2	Lists . . . . .	14
4.2	Python Library . . . . .	14
4.2.1	Random Function . . . . .	14
4.3	Gossip Protocol Designs . . . . .	15
4.3.1	ANY . . . . .	15
4.3.2	CO . . . . .	15
4.3.3	SPI . . . . .	15
4.3.4	LNS . . . . .	16
4.4	Gossip Simulator . . . . .	16
4.5	Simulation Runner . . . . .	17
<b>5</b>	<b>Implementation</b>	<b>18</b>
5.1	Agent Class Attributes ( <i>Fig 8.1</i> ) . . . . .	18
5.2	Agent Properties - Methods ( <i>Fig 8.1</i> ) . . . . .	18
5.3	Calculation of Calls - Agents and Received Messages ( <i>Fig 8.2</i> ) . . . . .	18
5.4	Gossip Protocols - Implementation . . . . .	19
5.5	Simulation Run <i>Fig(8.7)</i> . . . . .	20

5.6	Unit Tests and Results . . . . .	20
5.6.1	TestAgent Class ( <i>Fig 8.11</i> ) . . . . .	21
5.6.2	TestGossipSimulator Class . . . . .	21
5.6.3	Results . . . . .	22
<b>6</b>	<b>Evaluation</b>	<b>23</b>
6.1	Parameter Calculation(Formulas) . . . . .	23
6.2	Iterations for Final Analysis . . . . .	24
6.2.1	Iteration 1 . . . . .	24
6.2.2	Iteration 2 . . . . .	24
6.2.3	Iteration 3 . . . . .	24
6.2.4	Iteration 4 . . . . .	25
6.2.5	Average Results . . . . .	25
6.3	Visualisation of Results . . . . .	25
6.3.1	Average No. of Rounds Taken . . . . .	25
6.3.2	Average Contacts per Agent . . . . .	26
6.3.3	Total Number of messages known . . . . .	27
6.4	Critical Evaluation . . . . .	28
6.4.1	Advantages . . . . .	28
6.4.2	Shortcomings . . . . .	28
<b>7</b>	<b>Learning Points</b>	<b>29</b>
7.1	Acquired Skills and Knowledge . . . . .	29
7.2	Key Actions for Project Success . . . . .	29
7.3	Future Improvements . . . . .	29
7.4	Principal Learnings . . . . .	30
<b>8</b>	<b>Conclusion</b>	<b>31</b>
8.1	Reiteration of Goals and Targets . . . . .	31
8.2	Principal Discoveries . . . . .	31
8.3	Fulfilment of Goals and Targets . . . . .	31

# List of Figures

3.1	Fixed Fanout Gossip [1]	6
3.2	Probabilistic Edge Gossip[1]	7
3.3	Probabilistic Broadcast Gossip [1]	7
3.4	Probabilities	9
3.5	Relation between Message Complexity and Effectual Fanout	11
3.6	Difference between Fanout in GossipFF and Effectual Fanout	11
3.7	Any Call protocol	12
3.8	Call Once protocol	12
3.9	Learn New Secret (LNS)	13
3.10	Spider Method	13
6.1	Average number rounds of taken	26
6.2	Average Contacts per agent	27
6.3	Total messages known by protocols	27
8.1	Class Agent	34
8.2	Class GossipSimulator	34
8.3	Anycall function	34
8.4	Call Once function	35
8.5	Spider function	35
8.6	LearnNewSecret function	35
8.7	RunSimulation function	35
8.8	Unit test	36
8.9	Unit test 2	36
8.10	Unit test 3	36
8.11	Unit test 4	37



# List of Tables

6.1	Iteration 1-Summary of Protocols . . . . .	24
6.2	Iteration 2-Summary of Protocols . . . . .	24
6.3	Iteration 3-Summary of Protocols . . . . .	24
6.4	Iteration 4: Summary of Protocols . . . . .	25
6.5	Summary of Average Protocols . . . . .	25

# Chapter 1

## Introduction

Gossip Problem is where each individual in a group of agents holds a distinct secret, a gossip protocol is employed to enable the systematic exchange of these secrets through phone conversations. In every interaction, all the secrets that an agent has gathered until that point are exchanged.

The end goal of this procedure is to cultivate a group where every agent becomes an "expert," a term denoting an individual who has acquired all the secrets. This is achieved through a well-orchestrated communication protocol designed to ensure the full circulation of all secrets amongst all agents [2].

### 1.1 Scope

In the dynamic field of distributed systems, gossip protocols play a central role in enabling efficient peer-to-peer communication. These protocols are built upon the theoretical foundation of the gossip problem, where agents aim to share individual secrets across the network until every agent has access to all secrets. This project undertakes a detailed exploration of gossip protocols, focusing on well-established protocols including "ANY," "CO," "SPI," and "LNS."

### 1.2 Problem Statement

The core objective here is to deepen the understanding of gossip protocols, thereby enhancing the efficiency of information dissemination in distributed systems. This initiative is rooted in the imperative to scrutinize and compare the efficacy and reachability of various gossip protocols, a vital step towards fortifying peer-to-peer communications.

### 1.3 Approach

Utilizing Python, we crafted a Gossip Simulator to scrutinize the performance of different gossip protocols in a simulated setting. The simulator operates through a series of rounds where agents, each holding unique messages and following distinct rules, communicate until a specified condition of message dissemination is achieved or a maximum round limit is hit.

The simulator implements four distinct protocols, each outlining a different strategy for message sharing:

- **ANY Protocol:** Allows unrestricted communication between any agents [2]. *Example* , In the ANY protocol, imagine 5 friends each have a different secret. They can call any friend and share the secret

they know. Like, Alice tells Bob her secret, then Bob tells Charlie both his and Alice's secrets. Soon, by calling and sharing like this, all friends know all the secrets.

- **CO Protocol:** Restricts agents to a single communication event with another agent, preventing repeated exchanges[2]. *Example* , In this setup, imagine 5 buddies, each harboring a distinct secret. They are allowed to swap secrets just once. For instance, if Tom tells Jerry his secret, they can't exchange secrets anymore. However, Jerry can share Tom's secret with Spike. This ensures the secrets circulate without the same buddies sharing secrets repeatedly.
- **SPI Protocol:** Permits only token-holding agents to initiate communication, with the receiving agent losing its token after the interaction [2]. *Example*, In the SPI protocol, only kids holding a "special token" can start sharing secrets. Like, if Tim has a token, he can share a secret with Sara. After this, Sara loses her token, stopping her from sharing secrets first, but others can still share secrets with her. This controls how secrets are shared.
- **LNS Protocol:** Facilitates communication only if the receiving agent has at least one unknown message to the caller[2]. *Example* , In the LNS method, people only share news if the other person doesn't know it yet. Like, if Amy has news that Bob hasn't heard, she can tell him. But if Bob already knows it, Amy needs to find different news to share. This way, only new news gets passed around quickly.

## 1.4 Outcome

This project significantly contributes to the existing knowledge pool on gossip protocols by presenting a hands-on analysis through the simulation of various protocols. By offering empirical data on different protocols' performance metrics, it enables a detailed understanding of each protocol's strengths and weaknesses. Furthermore, the project offers graphical representations of the simulation results through bar graphs, facilitating an intuitive comprehension of the data. In essence, this project not only showcases the potential of gossip protocols in optimizing peer-to-peer communication but also sets the stage for future research in this arena. By establishing a foundational understanding and analysis, it invites further exploration into the creation of more efficient and robust protocols, fostering advancements in distributed systems and peer-to-peer communication networks. It is hoped that this project will guide both the academic and professional communities towards a deeper understanding, encouraging further innovations in this field.

## Chapter 2

# Aims and Objectives

### 2.1 Aims

- **Exploration of Gossip Protocols through Theoretical Analysis:** Delve into gossip protocols to theoretically discern critical parameters such as reachability, distribution strategies, randomization techniques, and patterns of information propagation. This analysis aims to foster a deeper comprehension of each protocol, paving the way for subsequent comparative analyses.
- **Development of a Simulator for the Comparative Study of Gossip Protocols:** Once the essential parameters are outlined, the next step is to craft a simulator to facilitate the comparative study of various gossip protocols. This involves establishing a simulation environment, delineating agent and network models, and instituting communication procedures.
- **Data Analysis to Derive Insightful Conclusions on Gossip Protocols:** Undertake a meticulous analysis of the amassed data to extract valuable insights into the efficacy and performance of different gossip protocols. This analysis aims to spotlight the strengths and shortcomings of each protocol, aiding informed decision-making in future endeavors.

### 2.2 Objectives

- **Recognition of Prevalent Gossip Protocols:** Embark on a research journey to pinpoint a collection of gossip protocols deemed to be potent solutions to the gossip problem. This entails understanding the tactics and methodologies leveraged by these protocols to facilitate the dissemination of information.
- **Formulation of Evaluation Criteria:** Set forth pertinent criteria to gauge the performance of the gossip protocols under scrutiny. These criteria should encompass aspects such as convergence duration, communication overhead, call frequency, and the effectiveness in realizing comprehensive information dissemination.
- **Simulator Construction:** Design a simulator that accurately replicates the conditions of the gossip problem, thereby allowing for the assessment and comparison of diverse gossip protocols. The simulator should be equipped to represent agents and their confidential information, establish communication pathways, and mimic the process of information exchange.

- **Establishment of a Simulation Environment:** Develop a simulation backdrop characterized by diverse network dimensions, agent setups, and secret allocations. This involves setting the parameters for the simulation, including network layout, initial information levels, and the conditions signaling the end of the simulation, to ensure uniform and regulated experiments.
- **Comparative Analysis Execution:** Execute simulations utilizing the integrated gossip protocols in the simulator. This involves a comparative analysis based on the predetermined evaluation criteria, assessing the efficiency in achieving full-scale information dissemination while weighing the balance between optimality and communication overhead.

## 2.3 Objectives Achieved

- **Gossip Protocol Identification:** The code recognizes and incorporates four distinct gossip protocols: ANY, CO, SPI, and LNS. Every protocol is characterized by its unique method.
- **Metric Definition for Evaluation:** The code outlines multiple metrics to gauge the efficacy of the gossip protocols: *Number of rounds*, *Mean interactions per agent*, *Aggregate messages acknowledged*. Yet, specifics like communication overhead and the efficiency of full information spread aren't directly addressed.
- **Simulator Creation:** The Gossip Simulator class in the code mirrors the gossip scenario. This simulator encompasses features to represent agents, their confidential data, communication routes, and the process of information sharing.
- **Environment for Simulation:** The code establishes a simulation setting with a designated count of agents (`num_agents`). Parameters such as *network structure*, *foundational knowledge*, and *end criteria* are indirectly set within the Gossip Simulator class's methods. Nonetheless, there's an absence of explicit differentiation or settings for diverse network dimensions, agent structures, and distribution of secrets.
- **Analysis and Comparison:** The code performs simulations employing the integrated gossip protocols. The efficacy of these protocols is ascertained using the predetermined evaluation metrics. Outcomes are saved in a textual file for subsequent scrutiny. Yet, an in-depth comparative study between optimal performance and communication expenses isn't present in the code.

# Chapter 3

## Background

### 3.1 Gossip Protocols

#### 3.1.1 Gossip Time Protocol (GTP)

GTP is a self-managing gossip time synchronization protocol designed for peer-to-peer networks. It has found applications in various fields, including non-traditional gossip applications such as gossip-based clock synchronization. The protocol operates in a network where each node is equipped with a local clock [3].

##### Operational Details

- **Presence of Accurate Node** - The protocol assumes the presence of at least one node in the network that maintains accurate and robust time information. This node serves as the primary source of time information, which is disseminated throughout the network.
- **Periodic Gossip** - Nodes in the network engage in periodic gossip sessions where they share their clock settings with randomly selected peers. This process ensures the propagation of accurate time information throughout the network.
- **Peer-Sampling Service** - GTP leverages a peer-sampling service that facilitates the selection of a uniformly random alive node for information exchange, enhancing the efficiency of the protocol.
- **Hop-Count Metric** - The quality of time at a node is determined based on a hop-count metric, which is essentially the number of nodes involved in the synchronization path from the time source to the node in question. The time source has a hop count of zero, and a node adopts a new time setting only if the hop count is larger than that of the communicating node plus one.
- **Gossiping Frequency** - Nodes can adapt their gossiping frequency based on their synchronization status. A well-synchronized node may choose to reduce its gossiping frequency, helping to stabilize the network over time.

##### Abstract Computing

In the GTP framework, "*Abstract Computing*" denotes the utilization of a hop-count metric to achieve time synchronization across a network, bypassing the need for exact clock offset computations between individual nodes[3]. This metric is grounded in the number of intermediary nodes (or "hops") a message traverses from the primary source of time information. Nodes will

only update their time settings based on information from a peer if it betters their hop count, thereby enhancing the precision of synchronization. This conceptual approach streamlines the synchronization procedure, centering on a relative gauge of time precision grounded in network structure, as opposed to exact time disparities, thus enabling streamlined and decentralized time harmonization in peer-to-peer networks.

### 3.1.2 GossipFF (Fixed Fanout Gossip)

This protocol is grounded on a set parameter termed "fanout," which establishes the count of designated neighbors to whom a message will be relayed during a gossip interaction.

- **Determined Recipient Count:** The strategy assures that a message reaches a predetermined count of neighbors, defined by the fanout parameter.
- **Enhanced Efficiency:** The strategy significantly diminishes the intricacies linked with message distribution, thereby boosting the efficiency of the communication mechanism.
- **Utilization:** It finds its significant application in expansive networks where overseeing message intricacy is essential to avert network bottleneck and sustain a steady information flow.

```

9 /* fanout: number of the selected
   neighbors */
10 GossipFF ( $\langle msg \rangle, fanout$ )
11   if  $fanout \geq V_i$  then
12      $toSend \leftarrow \Lambda_i$ 
13   else
14      $toSend \leftarrow \emptyset$ 
15     for  $f = 1$  to  $fanout$  do
16       random select  $s_j \in \Lambda_i / toSend$ 
17        $toSend \leftarrow toSend \cup s_j$ 
18   foreach  $s_j \in toSend$  do
19     Send( $\langle msg \rangle, s_j$ )

```

Figure 3.1: Fixed Fanout Gossip [1]

*Example,* In a corporate network of 1000 employees, a message needs to be quickly disseminated. Using the GossipFF protocol, each recipient is instructed to forward the message to 5 other randomly selected colleagues. This ensures a rapid spread of the message while controlling the traffic to prevent network congestion, facilitating efficient and orderly communication

### 3.1.3 GossipPE (Probabilistic Edge Gossip)

This protocol involves a probabilistic parameter that guides the selection of edges for the retransmission of a received message.

- **Selection Based on Chance:** The strategy involves choosing edges for message retransmission at random, guided by a predetermined probability parameter.

- Randomized Method: This method infuses randomness into message distribution, potentially averting network bottleneck and facilitating a broader message distribution over time.
- Suitability: It is apt for dynamic networks where the network structure is subject to change, and a probabilistic strategy can flexibly adapt to these alterations.

```

/*  $p_e$ : probability to use an edge */
GossipPE ( $\langle msg \rangle, p_e$ )
┌   foreach  $s_j \in \Lambda_i$  do
│   ┌   if Random()  $\leq p_e$  then
│   │   ┌   Send( $\langle msg \rangle, s_j$ )
│   │   └
│   └
└

```

Figure 3.2: Probabilistic Edge Gossip[1]

*Example,* In a social media platform, a user shares a post. The GossipPE protocol could be employed to probabilistically determine which connections (or "edges") in the user's network the post is shared with, based on a set probability parameter, ensuring a wide yet controlled dissemination of the post.

#### 3.1.4 GossipPB (Probabilistic Broadcast Gossip)

This protocol is based on a probability parameter that influences the decision to broadcast a message to all adjacent nodes.

- Stochastic Broadcasting: The broadcasting decision is influenced by a probability parameter, bringing a stochastic aspect to the broadcasting procedure.
- Equilibrium Between Reliability and Complexity: The strategy seeks to maintain a balance between assuring message reliability and minimizing message complexity to prevent network congestion.
- Suitability: It is optimal for networks aiming for a balanced communication approach, harmonizing reliability and efficiency.

```

/*  $p_v$ : probability to broadcast */
GossipPB ( $\langle msg \rangle, p_v$ )
┌   if Random()  $\leq p_v$  then
│   ┌   foreach  $s_j \in \Lambda_i$  do
│   │   ┌   Send( $\langle msg \rangle, s_j$ )
│   │   └
│   └
└

```

Figure 3.3: Probabilistic Broadcast Gossip [1]

*Example,* In a sensor network monitoring environmental conditions, a sensor detects a significant event and uses the GossipPB protocol to probabilistically decide whether to broadcast the alert to all other sensors in the network, aiming to maintain a balance between rapid alert dissemination and network resource conservation.



## 3.2 Gossip Protocols Analysis Approach

### Mean-Field Analysis Procedure

Mean-field analysis is a method utilized to scrutinize gossip protocols in expansive networks. This analysis is carried out through a series of defined stages[3], which are as follows-

**Step 1: Detailed System Outline -** Initially, a detailed and formal outline of the system or protocol that is being examined needs to be established. This assists in fostering a deeper comprehension and a segmented depiction of the system. It is necessary to formulate assumptions to streamline the system model, which should be validated through experimental research.

**Step 2: Determining Local States and Transition Paths -** At this juncture, it is vital to pinpoint the array of local states that a node in the network can assume. These states should encapsulate all the significant scenarios a node might encounter. The transitions between these states are predominantly facilitated through gossip exchanges.

**Step 3: Analysis of Transition Probabilities -** In this phase, a deep dive into the transition probabilities is undertaken. These probabilities, which are influenced by the universal state of the gossip network model, are meticulously examined. They dictate the potential of transitioning from one state to another, and the formulation of the transition probability matrix is reliant on the existing global state.

**Step 4: Setting the Criteria for Mean-Field Convergence -** To effectively apply mean-field analysis, it is imperative that the local transition probabilities exhibit suitable convergence as the node count tends towards infinity. This stipulation guarantees the applicability of the mean-field convergence theorem.

**Step 5: Establishing the Mean-Field Limit -** In the concluding step, the mean-field limit of the model is derived from the data acquired in the preceding stages. This limit sheds light on the functioning of the gossip protocol. The computation of the mean-field limit is facilitated through matrix-vector multiplications.

This structured approach delineates the systematic procedure for utilizing mean-field analysis in the evaluation of gossip protocols, guiding through each essential step in the process.

## 3.3 Keywords

### Transition Probabilities Explained

Transition probabilities, referred to as  $PN_{i,j}(m)$ , illustrate the likelihood of a node moving from one state,  $i$ , to another,  $j$ , based on the existing occupancy measure,  $m$ [3]. These probabilities are uniform across all entities in the system. The formulation of  $PN_{i,j}(m)$  involves several components. Initially, a node can instigate a gossip interaction, transitioning to state  $j$  by selecting a node  $B$  with a hop count of  $j-1$ . The probability of this event occurring is given by

$$\text{pull}N_{i,j}(m) = m_{j-1} \cdot N \cdot g_i \cdot \frac{1}{N-1} \cdot (1 - g_{j-1})$$

, where  $m_{j-1}$  denotes the proportion of nodes with a hop count equal to  $j-1$ .

Alternatively, a node might be approached by another node B, which is among the  $m_{j-1} \cdot N$  nodes with a hop count of  $j-1$ , without initiating gossip itself. The likelihood of this scenario is expressed as

$$\text{push}N_{i,j}(m) = m_{j-1} \cdot N \cdot g_{j-1} \cdot \frac{1}{N-1}$$

. These transition probabilities are pivotal in determining the dynamics of the gossip network model, playing a vital role in the mean-field analysis of gossip protocol performance.

## Mean field convergence

Mean-field convergence analysis is an essential tool in scrutinizing gossip protocols within expansive networks, removing the necessity for high-cost simulations. It facilitates researchers in delving deep into the scalability, dependability, and efficacy of these protocols in decentralized environments[3]. Scalability involves the protocol's capacity to sustain its functionality while handling an increasing number of users. Dependability ensures consistent and secure functioning in a network devoid of a central supervisory entity. Efficacy relates to achieving maximum output with minimal resource utilization. This analytical approach aids in assessing the protocols' resource management and communication facilitation, steering the creation of stronger and more efficient gossip protocols for widespread, decentralized network infrastructures. Essentially, mean-field convergence provides a budget-friendly avenue to scrutinize and improve the performance indicators of gossip protocols, encouraging progress in network systems.

## Mean Fields Limits

For  $N \rightarrow \infty$  we have the following limiting probabilities:

$$\begin{aligned} \text{pull}_{i,j}(\mathbf{m}) &= \lim_{N \rightarrow \infty} \text{pull}_{i,j}^N(\mathbf{m}) = g_i \cdot (1 - g_{j-1}) \cdot m_{j-1} \\ \text{push}_{i,j}(\mathbf{m}) &= \lim_{N \rightarrow \infty} \text{push}_{i,j}^N(\mathbf{m}) = g_{j-1} \cdot (1 - g_i) \cdot m_{j-1} \\ \text{noc}_{i,j}(\mathbf{m}) &= \lim_{N \rightarrow \infty} \text{noc}_{i,j}^N(\mathbf{m}) = e^{-\sum_{k=0}^{\infty} m_k \cdot g_k}. \end{aligned}$$

Figure 3.4: Probabilities

The discussion revolves around mean-field limits pertinent to gossip protocols, delineating the ultimate probabilities associated with various gossip interactions as the node count verges on infinity. These ultimate probabilities, categorized into pull, push, and no communication (noc) interactions, are denoted as  $\text{pull}_{i,j}(m)$ ,  $\text{push}_{i,j}(m)$ , and  $\text{noc}_{i,j}(m)$  respectively. They are derived by considering the limit of the respective probabilities for a finite node count as  $N$  tends towards infinity. The pull and push interactions' limiting probabilities are influenced by the gossip probabilities  $g_i$  and  $g_{j-1}$ , coupled with the hop count  $m$ . Conversely, the limiting probability for noc interactions is formulated through an exponential function, which incorporates the cumulative product of the hop count  $m$  and the gossip probability  $g_k$ , with  $k$  extending from 0 to infinity. These mean-field limits facilitate determining the limiting proportion of nodes characterized by a particular hop count at a specified moment. Leveraging this data enables the scrutiny and juxtaposition of diverse gossip protocol frameworks.

## 3.4 OMNeT++ Simulation

A feature of OMNeT++ is its approach to modeling. Unlike many network simulators that have a predetermined representation for network elements, OMNeT++ offers a generic component architecture. This flexibility allows model designers to represent concepts, be it network devices, protocols, or wireless channels, as they see fit. These model components, termed modules, are designed to be versatile, akin to LEGO blocks, and communicate primarily through message passing[4].

### 3.4.1 Parameters Used for Simulation

- **Fanout** refers to the count of messages a site forwards to its neighbours within a gossip algorithm, serving as an indicator of the site's message distribution capability.
- **Effectual Fanout** is a more comprehensive parameter. It not only considers the number of messages a site redistributes but also factors in the topology's degree distribution. This metric essentially gauges the average distribution strength of "infected" sites, meaning sites that have already received the message. There's a direct relationship between the effectual fanout and message complexity, making it a suitable metric for comparing gossip algorithms across varied topologies.

### 3.4.2 Topologies used for OMNeT++ Simulation

- **Bernoulli Graph  $B(N, p_N)$** : This type of graph is a stochastic bidirectional graph where connections between nodes are established randomly, each having a  $p_N$  probability of being connected, independent of other connections. Its design intends to create a substantial component encompassing  $N$  nodes, following a Poisson distribution for node degrees. The graph maintains a low edge dependency, implying that the presence of one edge is unrelated to the existence of others[3].
- **Random Geometric Graph  $G(N, \rho)$** : In this graph, nodes are distributed randomly within a confined space. A connection is formed between two nodes if their distance is less or equal to  $\rho$ . It features a Poisson distribution for node degrees and exhibits high edge dependency, indicating a correlation in the existence of its edges[5].
- **Scale-Free Graph  $S(N, m)$** : Generated through the Barabási-Albert model, this random bidirectional graph begins with a small clique of  $m_0$  nodes. As time progresses, new nodes are introduced, each connecting to  $m$  existing nodes in the graph. The likelihood of a new node connecting to an existing one is dictated by the existing node's degree, adhering to a principle of preferential attachment. This results in a network characterized by a power-law degree distribution, with distinct hub and peripheral nodes varying in degrees[6].

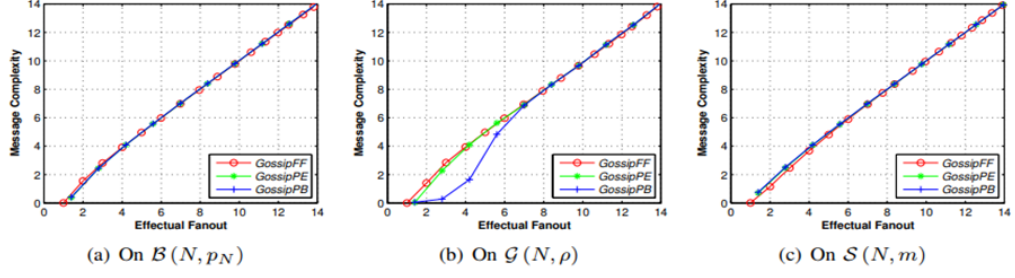


Figure 3.5: Relation between Message Complexity and Effectual Fanout

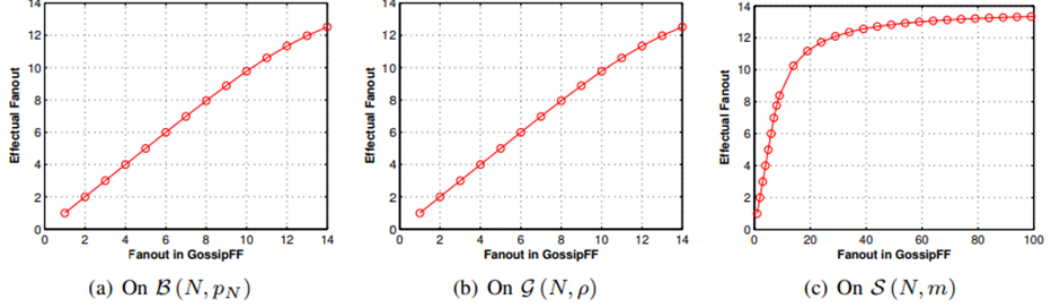


Figure 3.6: Difference between Fanout in GossipFF and Effectual Fanout

The graphs explain the difference between the fanout in GossipFF and the effectual fanout. The graph presented shows the value of the effectual fanout for each random topology as a function of the fanout in GossipFF. For small fanout values, there is an equality between the fanout and the effectual fanout. However, for large fanout values, the effectual fanout is proportionally smaller. This phenomenon occurs because sites with a smaller number of neighbors always retransmit fewer messages than the fanout value. In topologies with a high degree variance and a large number of sites with a small number of neighbors, this phenomenon is more pronounced. This explains why in many theoretical studies, the fanout of GossipFF is considered to be equal to or smaller than the minimum degree of the graph.

### 3.5 Implemented Protocols

- **ANY (Any Call Protocol):** The ANY protocol offers complete freedom for agents to communicate. Any agent can reach out to another, irrespective of their past interactions. This means there's no tracking of previous calls, making every interaction independent and fresh. *Explained Fig 3.7.*
- **CO (Call Once Protocol)** In the CO protocol, agents can only communicate once. After two agents have spoken, they can't converse again. This method avoids repetitive exchanges but might hinder complete information dissemination if not all agents get a chance to interact. *Explained Fig 3.8.*
- **LNS (Learn New Secrets Protocol)** The LNS protocol ensures agents only communicate when there's new information to share. An agent will call another only if they believe the other agent possesses unknown secrets. This approach guarantees that every conversation is meaningful and informative. *Explained Fig 3.9.*

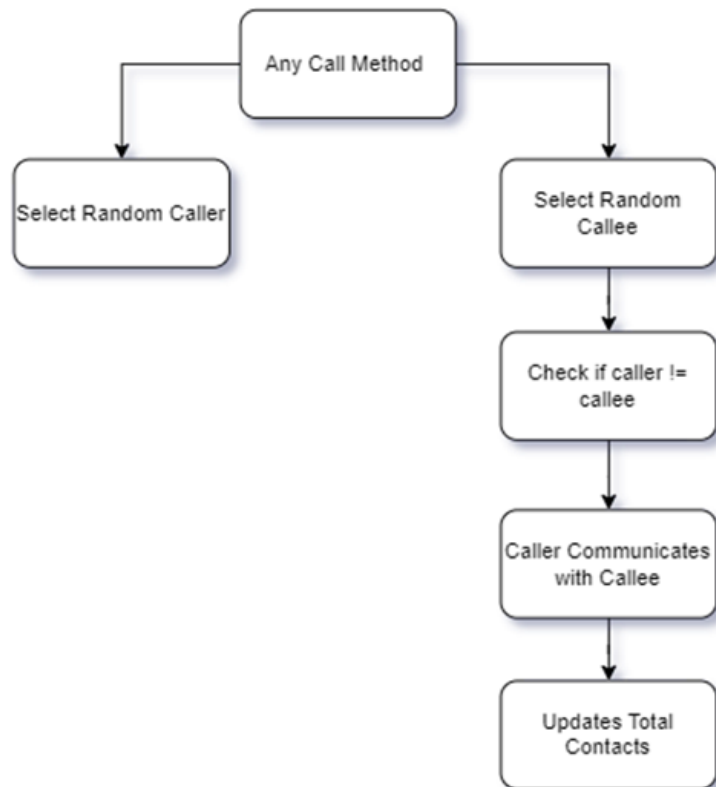


Figure 3.7: Any Call protocol

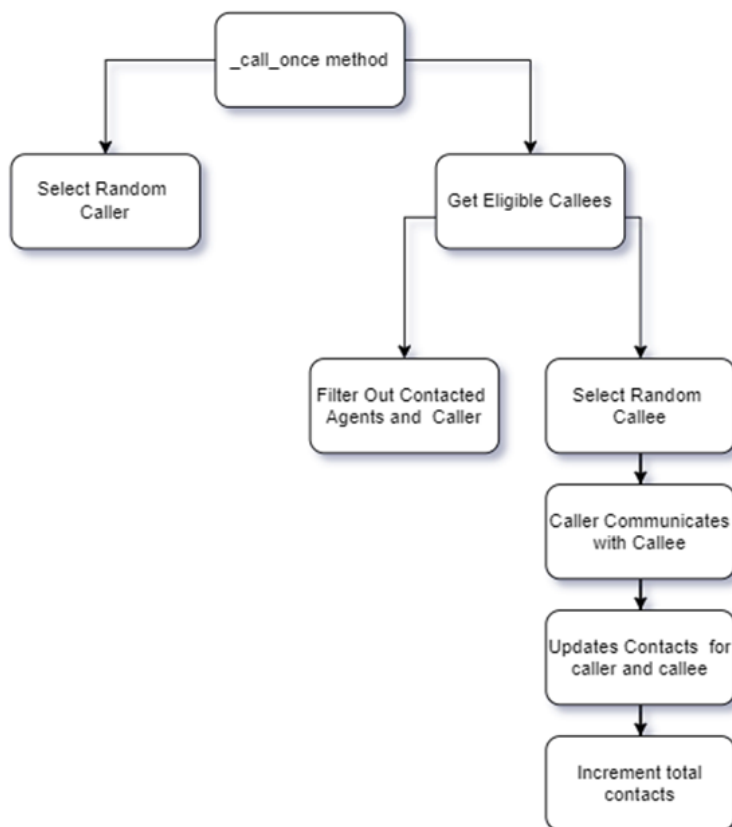


Figure 3.8: Call Once protocol

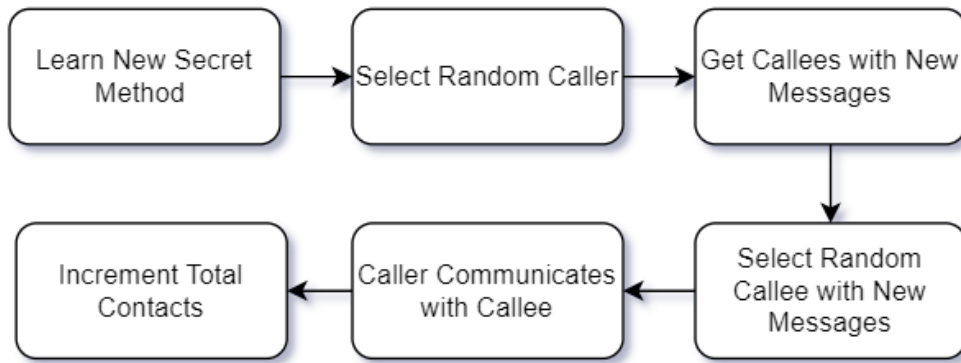


Figure 3.9: Learn New Secret (LNS)

- **SPI (Spider Protocol)** The SPI protocol is also token-based, but with a twist. Agents with tokens can call others, but after the call, the recipient gives their token to the caller. This means once an agent is called and loses their token, they can't initiate any more calls, limiting their future interactions. *Explained Fig 3.10.*

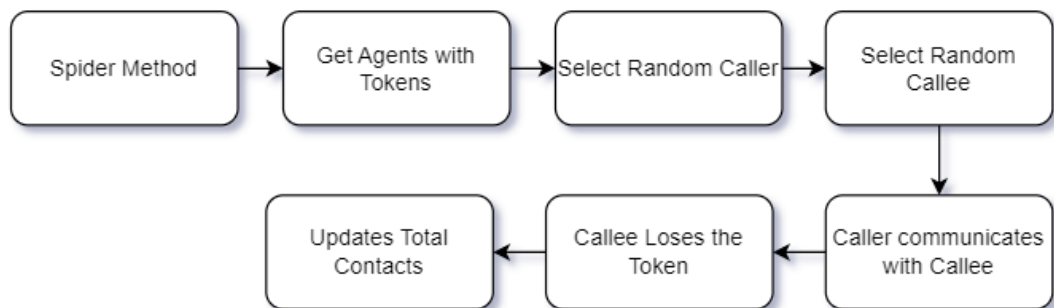


Figure 3.10: Spider Method

Each protocol sets distinct communication rules for agents, Detailed explanation of these protocols will be discussed in the next few chapters.

# Chapter 4

## Design

In this section, we delve into gossip networks where agents share information following set protocols, leveraging lists and sets for organized and efficient data management. We scrutinize different gossip protocols, examining their functioning and efficacy through specific metrics. The goal is to offer insights into realistic simulations of information dissemination in various networks.

### 4.1 Data Structures and Algorithms

#### 4.1.1 Sets

Sets in the simulation environment are utilized to store unique values, primarily to manage the messages known to each agent. Initially, an agent's set contains a message unique to them, identified by their unique ID. As agents communicate, they merge their sets of known messages, a process facilitated by the set data structure's ability to avoid duplication and allow efficient merging through union operations. This structure is also pivotal in the CO (Call Once) protocol, where it helps agents maintain a record of their past communications to prevent repeated exchanges, thus ensuring a one-time communication principle.

#### 4.1.2 Lists

The simulation leverages lists to organize and manage various aspects systematically. Initially, a list is created to house all agent entities, establishing a structured network for the simulation and allowing easy access to any agent based on its index. Lists facilitate the random selection of agents for interaction, a crucial aspect in simulating spontaneous communications. Moreover, they are used to accumulate the results of each simulation cycle, with each entry in the list being a detailed account of an individual run, thereby simplifying the analysis and summarization of results post-simulation. Lists also aid in gathering agents who meet specific criteria for interaction in certain protocols, streamlining the selection process from a pool of qualified agents.

### 4.2 Python Library

#### 4.2.1 Random Function

The random function from the imported library plays a crucial role in enhancing the stochastic nature of the simulation, introducing an element of unpredictability in agent interactions. It is employed in various protocols

to randomly select agents for conversations, a strategy that mirrors real-life spontaneous interactions and creates dynamic and unforeseeable communication pathways. This randomness is central to evaluating different gossip protocols' efficacy and productivity. Furthermore, it facilitates the implementation of diverse protocols by aiding in the random selection of agents based on different criteria, thus maintaining the simulation's stochastic essence and providing a realistic representation of gossip dissemination in a network. The random selections foster a natural and spontaneous interaction environment, ensuring a wide range of potential outcomes and offering a detailed perspective on different protocols' performance in various scenarios, aiding in the analysis of their resilience and dependability.

## 4.3 Gossip Protocol Designs

### 4.3.1 ANY

In the designated segment of the simulation, the "ANY" protocol allows for unrestricted communication between different entities within a network. Essentially, any individual can engage in an exchange of information with another individual chosen at random, provided they are not attempting to communicate with themselves. This ensures a genuine transfer and acquisition of knowledge. During this process, the system keeps track of the total number of successful exchanges, incrementing a counter to record the dynamics of information flow within the network. This method encourages spontaneous and natural dialogues, akin to real-life scenarios where interactions are not bound by any predetermined rules, thereby fostering a dynamic and organic pathway for the dissemination of information. It aims to realistically portray the randomness and spontaneity seen in natural communication processes, where there are no fixed patterns dictating the flow of conversations.

### 4.3.2 CO

In the "CO" or "Call Once" protocol, the system is designed to facilitate a one-time communication between two distinct entities in the network. To achieve this, an initiating entity is randomly selected to communicate with another entity, chosen from a pool of individuals it has not previously interacted with. This ensures that each pair of entities communicates only once throughout the simulation. After a successful interaction, the system updates the records for both entities to note this communication, preventing any future repetitions. Moreover, the overall count of interactions within the network is increased to keep track of the total communications. This protocol is designed to mimic scenarios where each interaction is unique, avoiding repeated exchanges and promoting a diverse and expansive spread of information through one-off communications.

### 4.3.3 SPI

In this specific strategy, there is a group of nodes in a network that have the special privilege, denoted by a token, to initiate interactions. During each interaction cycle, one node holding a token is chosen at random to interact with another randomly selected node from the entire network, ensuring that the two nodes are not the same.

Upon the completion of the interaction, the node on the receiving end of the communication has to forfeit their token, thereby losing their spe-



cial privilege to initiate future interactions. This mechanism ensures a dynamic where the number of nodes that can start interactions diminishes over time, steering the communication process in a unique direction. Moreover, a counter that keeps track of the total number of interactions taking place is incremented with each successful interaction, helping in monitoring the overall activity level in the network. This strategy creates a scenario where the communication dynamics evolve with each interaction, with a decreasing number of nodes being able to initiate conversations as time progresses.

#### 4.3.4 LNS

In the designed strategy, a network is formed where individual nodes have the ability to communicate based on the uniqueness of the information they hold. During a communication cycle, a node is randomly selected to initiate a conversation. This node doesn't communicate with just any other node in the network; it specifically seeks out nodes that possess at least one piece of information that it hasn't encountered before, ensuring that every interaction is meaningful and results in the acquisition of new knowledge.

To identify a suitable partner for communication, the initiating node scans the network to find others holding at least one unique data point that it is unaware of. Once identified, the two nodes engage in a knowledge exchange, enhancing the information repository of the initiating node. This method encourages a network dynamic where every communication is beneficial, promoting a continuous enrichment of knowledge as nodes are always in the pursuit of new information. The system keeps a tally of the total number of successful interactions, helping in monitoring the spread of information throughout the network over time.

**Designing an Agent** In gossip networks, an "agent" is a unique entity equipped to send and receive messages. It has distinct attributes including a unique identifier for differentiation, a collection of known messages that grows over time, and a "token" granting communication privileges, which can be withdrawn based on protocol rules. Additionally, it keeps a contact log to avoid repeated interactions, ensuring protocol adherence and efficient communication.

### 4.4 Gossip Simulator

The Gossip Simulator serves as the nucleus of the network, facilitating interactions between agents based on established rules and protocols. It is characterized by a set of attributes: the defined number of agents which delineates the network's scope, and a cap on the simulation rounds to avoid infinite loops, ensuring a controlled environment for the simulation. The simulator harbors a list of all agent entities and maintains a tally of all the interactions that transpire during the simulation. It is endowed with a range of functionalities, including a method that empowers agents to exchange known messages and another that allows them to forfeit their right to initiate communications under stipulated protocols. Furthermore, it possesses a method to access the knowledge state of each agent at any juncture in the simulation, offering a dynamic overview of the evolving information landscape. The simulator is also equipped with a suite of methods designed to enact various communication protocols, thereby serving as the foundational framework that guides the simulation process.

## 4.5 Simulation Runner

The Simulation Runner acts as the orchestrator, initializing the Gossip Simulator with precise parameters and steering the simulation through different protocols while accumulating and archiving the results for subsequent analysis. It is defined by several pivotal attributes, including housing an instance of the Gossip Simulator, which is the driving force behind the simulation operations. It also delineates a list of protocols slated for testing, facilitating a multifaceted examination of the gossip network. Moreover, it oversees a repository to chronicle the outcomes of each simulation iteration, ensuring a structured data compilation for analytical purposes. Additionally, it documents the message knowledge of each agent post every simulation, providing a detailed account of information dissemination. While devoid of its own methods, it leverages the functionalities present in the Gossip Simulator to execute the simulations, playing an indispensable role in realizing the simulation goals.

## Chapter 5

# Implementation

### 5.1 Agent Class Attributes (*Fig 8.1*)

- **id**: A unique identifier for the agent. It is set during the initialization of an agent object.
- **known\_messages**: A set that contains messages known to the agent. Initially, it contains only the agent's own ID. These IDs will be used to communicate while interacting with other agents.
- **has\_token**: A boolean flag indicating whether the agent has a token. Initially, every agent has a token.
- **contacts**: A set that keeps track of the agents that this agent has communicated with. Initially, it is empty.

### 5.2 Agent Properties - Methods (*Fig 8.1*)

- **\_\_init\_\_(self, id, total\_agents)**: The constructor method which initializes a new agent object with the given **id** and **total\_agents**. Note that the **total\_agents** parameter is not used in the initialization process; it seems to be a redundant parameter.
- **communicate(self, other)**: This method allows the agent to communicate with another agent (**other**). During the communication, both agents exchange all the messages they know, updating their **known\_messages** sets accordingly.
- **remove\_token(self)**: This method sets the **has\_token** attribute of the agent to **False**, indicating that the agent no longer has a token.

### 5.3 Calculation of Calls - Agents and Received Messages (*Fig 8.2*)

```
def __init__(self, num_agents, max_rounds=10000):
```

This is the constructor method for the Gossip Simulator class. It takes two parameters: **num\_agents**, which is the number of agents in the simulation, and **max\_rounds**, which is the maximum number of rounds the simulation will run for. The **max\_rounds** parameter has a default value of 10000. The value is kept at 10000 to avoid the infinite loop, as there could be a case where even after an infinite number of contacts the agent won't become an expert. The Gossip Simulator class is crafted to emulate a gossip protocol involving a predetermined number of agents and a maximum limit on the

rounds of communication. It sets up a roster of agents and offers a function to fetch the existing status of messages acknowledged by each agent. While it maintains a record of the cumulative number of contacts or interactions between agents, this feature is presently dormant and not utilized in any existing method, serving as a reserve for possible upcoming features to tally the communication instances.

## 5.4 Gossip Protocols - Implementation

### 1. `_any_call` Function (*Fig 8.3*)

#### Description:

- **Protocol:** ANY
- **Functionality:** In this function, any agent can communicate with any other agent in the system.
- **Implementation:**
  - **Caller Selection:** An agent is randomly chosen from the list of agents to be the caller.
  - **Callee Selection:** Another agent is randomly chosen from the list of agents to be the callee.
  - **Communication:** If the caller is not the same as the callee, a communication is established between them through the `communicate` method of the caller object.
  - **Total Contacts:** The total number of contacts is incremented by 1.

### 2. `_call_once` Function (*Fig 8.4*)

#### Description:

- **Protocol:** CO (Call Once)
- **Functionality:** An agent can communicate with another agent only once.
- **Implementation:**
  - **Caller Selection:** An agent is randomly chosen to be the caller.
  - **Eligible Callees:** A list of eligible callees is created, excluding agents that have already been contacted by the caller and the caller itself.
  - **Communication:** If there are any eligible callees (DEFINE), one is chosen randomly and communication is established.
  - **Contact Update:** The sets of contacts for both the caller and the callee are updated to include each other's ID.
  - **Total Contacts:** The total number of contacts is incremented by 1.

### 3. `_spider` Function (*Fig 8.5*)

#### Description:

- **Protocol:** SPI (Spider)
- **Functionality:** Only agents holding a token can initiate a call. The callee loses its token upon being contacted.
- **Implementation:**

- **Eligible Callers:** A list of eligible callers is created, including only agents that hold a token.
- **Callee Selection and Communication:** If there are eligible callers, one is chosen randomly along with a random callee, and communication is established if they are not the same.
- **Token Removal:** The callee loses its token after being contacted.
- **Total Contacts:** The total number of contacts is incremented by 1.

#### 4. `_learn_new_secrets` Function (*Fig 8.5*)

##### Description:

- **Protocol:** LNS (Learn New Secrets)
- **Functionality:** An agent calls another agent only if the callee possesses at least one message unknown to the caller.
- **Implementation:**
  - **Caller Selection:** An agent is randomly chosen to be the caller.
  - **Eligible Callees:** A list of eligible callees is created, including agents that know at least one message that the caller does not.
  - **Communication:** If there are any eligible callees, one is chosen randomly and communication is established.
  - **Total Contacts:** The total number of contacts is incremented by 1.

Each function implements a different protocol for agent communication, manipulating the `total_contacts` attribute to keep track of the total number of communications that have occurred. They all use random selection for choosing callers and callees, and they facilitate communication through the `communicate` method of the agent objects.

## 5.5 Simulation Run *Fig(8.7)*

The `'run_simulation'` method conducts a gossip simulation based on a chosen protocol. It continues until all agents have the complete information or a set number of rounds is reached. Initially, a `'rounds'` counter is set to zero. The simulation then checks the protocol type and triggers the relevant method: `'_any_call'` for "ANY", `'_call_once'` for "CO", `_spider` for "SPI", and `'_learn_new_secrets'` for "LNS". With each iteration, the `'rounds'` counter increases. After the loop, performance metrics are computed. The average contacts each agent has is calculated by dividing total contacts by the agent count. Additionally, the cumulative messages known by all agents are summed up. To ensure a fresh start for future simulations, the agent list and total contacts are reset.

## 5.6 Unit Tests and Results

The provided code offers a suite of unit tests for the Gossip Simulator program. These tests are crucial for validating the program's functionality, ensuring that both the Agent and GossipSimulator classes operate as expected.

### 5.6.1 TestAgent Class (*Fig 8.11*)

- **setUp Method** The `setUp` method in the `TestAgent` class prepares the testing environment for the `Agent` class. It initializes a new agent before each test, ensuring a consistent starting point for every test scenario.
- **test\_init Method** The `test_init` method evaluates the initialization process of the `Agent` class. It checks that each agent is correctly assigned an ID, starts with knowledge of only its own message, possesses a token, and has no initial contacts. This method ensures that each agent starts with a consistent and correct state.
- **test\_communicate Method** The `test_communicate` method assesses an agent's ability to exchange messages with another agent. By simulating a communication event between two agents, this test ensures that both agents update their known messages correctly, reflecting the shared information.
- **test\_remove\_token Method** The `test_remove_token` method examines an agent's token management. After invoking the method to remove a token, the test checks that the agent no longer possesses a token. This ensures that the simulator can correctly manage and update an agent's token status.

### 5.6.2 TestGossipSimulator Class

- **setUp Method** The `setUp` method initializes the testing environment. Before each test, a new `GossipSimulator` instance is created with three agents. This setup ensures that every test starts with a consistent and fresh state, eliminating potential biases from previous tests (*Fig 8.8*).
- **test\_init Method** The `test_init` method scrutinizes the initialization process of the `GossipSimulator` class. It verifies that the simulator correctly sets the number of agents, initializes the maximum rounds to its default, creates the right number of agent objects, and starts with zero total contacts. This method ensures the foundational settings of the simulator are accurate (*Fig 8.8*).
- **test\_get\_agent\_messages Method** The `test_get_agent_messages` method evaluates the simulator's ability to track messages known by each agent. Initially, every agent should only be aware of its own message. This test confirms that the simulator correctly maintains and retrieves this information, ensuring the integrity of the gossip process (*Fig 8.9*).
- **test\_any\_call Method** The `test_any_call` method assesses the "ANY" gossip protocol's functionality. In this protocol, any agent can communicate with another. The test ensures that after a single communication event, the simulator correctly updates the total contacts, reflecting the interaction between agents (*Fig 8.9*).
- **test\_call\_once Method** The `test_call_once` method delves into the "CO" gossip protocol. Here, an agent communicates with another agent only once. The test checks that the simulator accurately updates the total contacts after such a communication, ensuring the protocol's unique rules are upheld (*Fig 8.9*).

- **test\_spider Method** ) The `test_spider` method evaluates the "SPI" gossip protocol. In this scenario, only agents with tokens can initiate communication. Upon being contacted, the receiving agent loses its token. This test ensures that the simulator correctly manages tokens and updates total contacts in line with the protocol's rules(*Fig 8.9*).
- **test\_learn\_new\_secrets Method** The `test_learn_new_secrets` method examines the "LNS" gossip protocol. Here, an agent communicates with another only if the latter has unknown messages. This test simulates a scenario where one agent has an unfamiliar message and verifies that the simulator correctly updates the total contacts after the communication (*Fig 8.10*).
- **test\_run\_simulation Method** The `test_run_simulation` method is a comprehensive test for the entire simulation process. It runs the simulation for the "ANY" protocol and checks the results. The test ensures that the simulator correctly logs the protocol name, rounds taken, average contacts per agent, and total messages known, validating the simulation's overall functionality (*Fig 8.10*).

### 5.6.3 Results

All the unit test passed Successfully

## Chapter 6

# Evaluation

### 6.1 Parameter Calculation(Formulas)

- **Rounds Taken :** Rounds taken refers to the number of cycles needed for all individuals (agents) in a group to receive all the messages, or until a preset maximum number of cycles is reached. In each cycle, agents share messages according to specific rules of the gossip protocol being used. The cycle count starts at zero and increases by one with each cycle until the end conditions are met.
- **Average Contacts per Agent :** The formula to calculate the average contacts per agent is given by:

$$\text{avg\_contacts} = \frac{\text{self.total\_contacts}}{\text{self.num\_agents}}$$

where:

- **avg\_contacts** is average number of contacts per agent.
- **self.total\_contacts** is the total number of contacts made during the simulation
- **self.num\_agents** is the total number of agents in the simulation
- **Total messages known :** The total messages known is calculated using the following formula:

$$\text{total\_messages\_known} = \sum_{i=1}^{\text{self.num\_agents}} \text{len}(\text{self.agents}[i].\text{known\_messages})$$

where:

- **total\_messages\_known** is the total number of messages known to all agents
- **self.agents[i].known\_messages** is the list of messages known to agent  $i$
- **len(.)** is a function that returns the number of elements in a list
- The sum runs over all agents in the simulation



## 6.2 Iterations for Final Analysis

### 6.2.1 Iteration 1

Table 6.1: Iteration 1-Summary of Protocols

Protocol	Rounds taken	Average contacts per agent	Total messages known
ANY	33	4.0	49
CO	18	2.571	49
SPI	38	4.143	49
LNS	19	2.0	49

In the first iteration, we observe the performance of each protocol in a single iteration of the simulation. The "SPI" protocol took the highest number of rounds (38) to ensure that all agents know all the messages, while the "CO" protocol was the most efficient with only 18 rounds. Despite the differences in rounds and average contacts per agent, all protocols successfully ensured that all messages were known to all agents by the end of the simulation.

### 6.2.2 Iteration 2

Table 6.2: Iteration 2-Summary of Protocols

Protocol	Rounds taken	Average contacts per agent	Total messages known
ANY	21	2.714	49
CO	19	2.571	49
SPI	37	4.571	49
LNS	22	2.143	49

In the second iteration, we notice a change in the dynamics where the "SPI" protocol still took a higher number of rounds compared to others, but the "LNS" protocol improved significantly, requiring only 22 rounds. This run showcases the variability in the number of rounds and contacts per agent that can occur in different iterations of the simulation, emphasizing the stochastic nature of the process.

### 6.2.3 Iteration 3

Table 6.3: Iteration 3-Summary of Protocols

Protocol	Rounds taken	Average contacts per agent	Total messages known
ANY	27	3.286	49
CO	23	2.714	49
SPI	29	3.857	49
LNS	15	2.143	49

The third iteration presents a scenario where the "LNS" protocol performed exceptionally well, requiring the least number of rounds (15) to disseminate all the messages to all agents. This run highlights the potential efficiency of the "LNS" protocol under certain conditions. It also shows that the "ANY" protocol had a moderate performance with 27 rounds.

### 6.2.4 Iteration 4

Table 6.4: Iteration 4: Summary of Protocols

Protocol	Rounds taken	Average contacts per agent	Total messages known
ANY	16	2.0	49
CO	16	2.286	49
SPI	35	4.0	49
LNS	28	2.0	49

In the fourth iteration, both the "ANY" and "CO" protocols performed equally well in terms of rounds taken, which was the lowest among all protocols in this run. This indicates that in some instances, simpler protocols can be as efficient as or even outperform more complex ones. The "SPI" protocol, however, remained consistent in taking a higher number of rounds in each run.

### 6.2.5 Average Results

Table 6.5: Summary of Average Protocols

Protocol	Avg. Rounds taken	Avg. contacts per agent	Avg. Total msg known
ANY	24.25	3.0	49
CO	19.0	2.536	49
SPI	34.75	4.143	49
LNS	21.0	2.071	49

The average results table amalgamates the data from all individual runs to provide a general overview of the performance of each protocol over multiple simulations. Here, we can see that, on average, the "CO" protocol was the most efficient, requiring the least number of rounds and having a lower average contact per agent. The "SPI" protocol, on the other hand, took the most rounds on average to complete the message dissemination.

Despite the differences in the number of rounds and contacts per agent, all protocols were successful in ensuring that all messages were known to all agents, as indicated by the constant "Average Total messages known" value of 49 across all protocols and runs.

## 6.3 Visualisation of Results

### 6.3.1 Average No. of Rounds Taken

This graph clearly depicts the average rounds taken by each protocol, facilitating a side-by-side comparison of their performances across trials. It highlights that the SPI protocol typically necessitated more rounds, implying extended trial durations, whereas the CO protocol demonstrated efficiency with fewer rounds on average. This visualization is crucial in analyzing the overall efficacy and time management of each protocol, assisting in a detailed comparative study of their trial performances. It aids in understanding the varying complexities and time frames associated with each protocol.

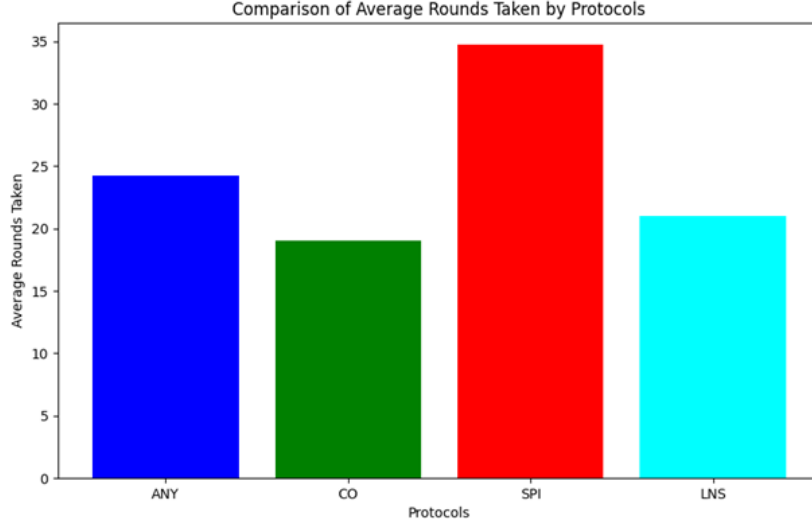


Figure 6.1: Average number rounds of taken

### Analysis

- **ANY Protocol:** The ANY protocol has a moderate average number of rounds taken, indicating a balanced performance over the trials.
- **CO Protocol:** The CO protocol showcases a slightly lower average number of rounds taken, suggesting a more efficient process in terms of rounds utilized across the trials.
- **SPI Protocol:** The SPI protocol stands out with the highest average number of rounds taken, pointing to a more complex process with a higher number of rounds needed to complete the trials.
- **LNS Protocol:** The LNS protocol exhibits a moderate average number of rounds taken, which is somewhat in the middle range compared to other protocols, indicating a balanced approach in the trial process.

### 6.3.2 Average Contacts per Agent

This graph succinctly outlines the average contacts per agent for each protocol, enabling a side-by-side evaluation of their performances during the trials. It unveils that the SPI protocol encouraged the highest level of interaction, whereas the LNS protocol favored limited engagements. The graph stands as a pivotal tool in comprehending the interaction tendencies of each protocol, assisting in identifying the collaboration intensity and communication complexity prevailing in the trials of different protocols.

### Analysis

- **ANY Protocol:** The ANY protocol registers a higher mean contacts per agent, signaling a collaborative and interactive backdrop throughout its trials.
- **CO Protocol:** The CO protocol maintains a balanced level of average contacts per agent, indicating a harmonious interaction landscape across its trials.

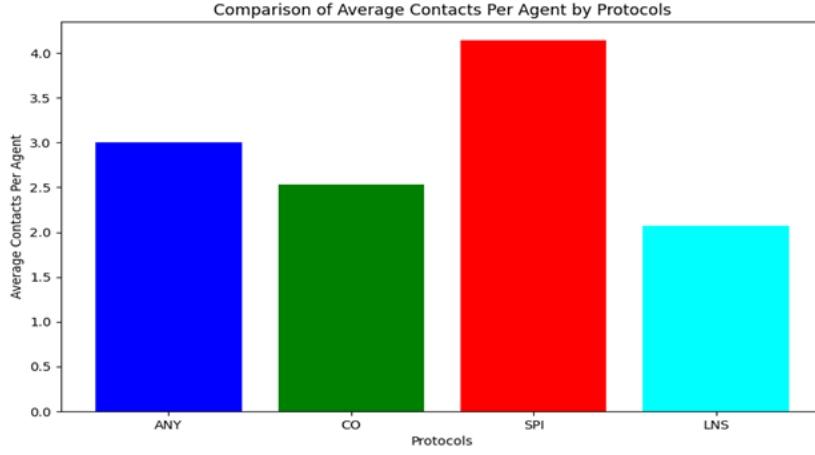


Figure 6.2: Average Contacts per agent

- **SPI Protocol:** The SPI protocol emerges with the peak average contacts per agent, denoting a vibrant and dynamic interaction sphere during its trials.
- **LNS Protocol:** The LNS protocol records the minimum average contacts per agent, hinting at a more linear and less interactive communication pattern compared to its counterparts.

### 6.3.3 Total Number of messages known

The graph succinctly delineates that the knowledge foundation, in terms of total messages known, remained unchanged across all protocols, each exhibiting an average of 49 known messages. This reveals a harmony in the knowledge base across different protocols, signifying that the information regarding messages was uniformly acknowledged in each protocol setting. It acts as an instrument to comprehend the uniform distribution of knowledge across diverse protocols

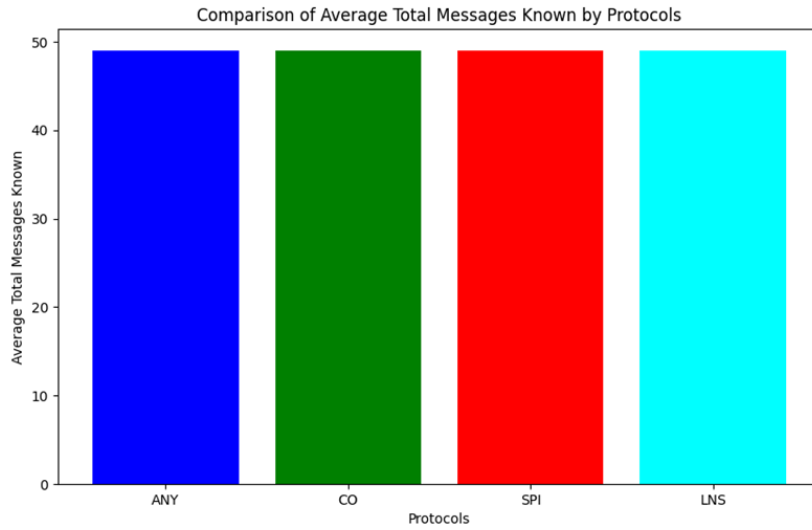


Figure 6.3: Total messages known by protocols

## Analysis

The graph distinctly illustrates that the average total messages known is uniformly 49 for all protocols, showcasing a stable knowledge foundation across the various protocols

## 6.4 Critical Evaluation

### 6.4.1 Advantages

- **Adaptability:** The simulation tool is adaptable, endorsing the straightforward integration or alteration of protocols to scrutinize diverse dynamics in the gossip mechanism.
- **Stochastic Simulation:** The incorporation of stochastic elements in agent selection imparts a degree of realism to the simulation, mirroring the spontaneity found in real-world contexts more faithfully.
- **Data Management:** The initiative ensures the meticulous recording and subsequent retrieval of simulation data, fostering data longevity and uncomplicated accessibility.
- **Graphical Depiction:** The project leverages graphical illustrations to succinctly portray performance metrics, thereby simplifying data analysis and elucidation.

### 6.4.2 Shortcomings

- **Restricted Error Mitigation:** The existing framework exhibits a deficiency in error management, particularly when extracting data from the text document, a factor that could precipitate complications in instances of file modification or corruption.
- **Static Agent Count:** The simulations are executed with a predetermined agent count, a constraint that potentially restricts insights into the protocols' functionality under varying agent populations.
- **Recurrent Initialization:** The current setup necessitates the repeated initialization of agents following each simulation cycle, a process that could augment computational demands and decelerate simulations involving a substantial agent pool or round allocation.

# Chapter 7

## Learning Points

### 7.1 Acquired Skills and Knowledge

- **Simulation Crafting:** Gained insights into crafting a simulation environment where agents communicate based on distinct protocols.
- **Utilization of Random Module:** Utilized Python's random module to facilitate random selections, a pivotal element in the gossip simulation.
- **Analytical Skills:** Developed skills in dissecting the simulation results to gauge the efficacy of various protocols.

### 7.2 Key Actions for Project Success

- **Protocol Definition:** The success was partly due to the well-defined protocols (ANY, CO, SPI, LNS) with unique rules guiding agent interactions.
- **Metric Implementation:** Implemented pertinent metrics such as "rounds taken," "average contacts per agent," and "total messages known" to assess the protocols.
- **Multiple Simulations:** Undertook numerous simulation runs to amass a rich dataset for analysis.
- **Agent Reset:** Ensured the integrity of each simulation by resetting agents and contact counters post each run.

### 7.3 Future Improvements

- **Incorporating Error Handling:** In future, would add error handling to deal with potential issues during the simulation, enhancing code robustness.
- **Code Optimization:** Would seek avenues to enhance code performance, possibly through the adoption of efficient data structures or algorithms.
- **Flexible Adjustments:** Plan to add features for dynamic alterations to the number of agents and maximum rounds, facilitating a broader simulation scope.

- **Data Visualization:** Aim to incorporate tools for graphical representation of results, offering a more intuitive understanding of the outcomes.
- **Extensive Testing:** Intend to conduct rigorous testing to identify and fix bugs, ensuring the accurate depiction of protocols in the simulation.

## 7.4 Principal Learnings

- **Protocol Efficiency:** Recognized that protocols vary in efficiency, with some necessitating fewer rounds and contacts per agent to circulate all messages.
- **Impact of Random Selection:** Acknowledged that random agent selection for communication can cause significant result variations in different simulation runs.
- **Simpler Protocols' Potential:** Realized that simpler protocols might sometimes surpass complex ones, indicating a potential area for further study and optimization.
- **Effective Message Spread:** All protocols ensured complete message dissemination among agents, showcasing the efficacy of gossip protocols in information spread.
- **Real-World Relevance:** The project's learnings find applications in real-world scenarios, especially in network communications and information dissemination strategies.

# Chapter 8

## Conclusion

### 8.1 Reiteration of Goals and Targets

As mentioned in section 2.1 The initiative commenced with an in-depth theoretical scrutiny of gossip protocols, aiming to identify vital parameters such as reachability, distribution strategies, and the dynamics of information flow. The goal was to cultivate a deep comprehension of each protocol, thereby laying the groundwork for subsequent comparative analyses.

Following this, we engineered a simulator as seen in Chapter 5 and its sections to aid in the comparative study of various gossip protocols. This tool defined agent which is implemented as seen in section 5.2 and network models, created a simulation environment, and set up communication processes as discussed in section 5.3, enabling a detailed analysis of the gathered data to glean crucial insights into the effectiveness and operational dynamics of different gossip protocols.

### 8.2 Principal Discoveries

The simulation outcomes, generated from a Python script that utilized classes and functions to depict agents and their intercommunications based on distinct protocols, offered a substantial dataset for examination as we have seen in Chapter 6 Section 6.2. The protocols examined were ANY, CO, SPI, and LNS, each governed by unique rules dictating agent communications.

Upon scrutinizing the results, it was evident that all the protocols adeptly enabled message dissemination among agents, albeit with differing efficiencies. The SPI protocol generally necessitated more rounds, indicating a greater communication burden. On the other hand, the LNS protocol demonstrated efficiency, accomplishing complete message dissemination in fewer rounds during certain runs that can be seen in the visual representation of the results in section 6.3.

The mean contacts per agent exhibited considerable variation across protocols and runs, with the ANY protocol demonstrating a propensity for higher average contacts, underscoring a more assertive message dissemination strategy. The CO protocol adopted a balanced stance, harmonizing communication overhead with efficacy.

### 8.3 Fulfilment of Goals and Targets

We have successfully attained the goals and targets delineated at the onset of the project. The theoretical scrutiny of gossip protocols was exhaustive,



facilitating the creation of a simulator that precisely mirrored the conditions of the gossip dilemma. The simulator proficiently represented agents and their confidential data, forging communication routes and simulating the information exchange process, thereby achieving the target of simulator development.

The simulation backdrop was crafted with a variety of network dimensions, agent configurations, and secret distributions, satisfying the target of crafting a suitable simulation environment for the experiments. The comparative analysis was meticulously executed, evaluating the efficacy of each protocol in facilitating comprehensive information dissemination while considering the equilibrium between optimality and communication overhead, thus meeting the target of executing a comparative analysis. Prospects for Future Endeavours

While the project has met its primary targets, it unveils opportunities for further investigations. Subsequent endeavours could encompass:

- **Advanced Simulation:** Introducing more intricate network configurations and agent dynamics to mirror more realistic circumstances.
- **Protocol Optimization:** Further exploring each protocol to identify optimization avenues, enhancing performance without increasing the communication burden.
- **Real-World Implementations:** Investigating the practical applications of these protocols in network communications and information distribution strategies, potentially finding resolutions to prevailing challenges in these areas.

# Bibliography

- [1] Ruijing Hu, Julien Sopena, Luciana Arantes, Pierre Sens, and Isabelle Demeure. Fair comparison of gossip algorithms over large-scale random topologies. *2012 IEEE 31st Symposium on Reliable Distributed Systems*, 2012.
- [2] Hans van Ditmarsch, Malvin Gatteringer, Ioannis Kokkinis, and Louwe B. Kuijer. Reachability of five gossip protocols. *Lecture Notes in Computer Science*, page 218–231, 2019.
- [3] Rena Bakhshi, Lucia Cloth, Wan Fokkink, and Boudewijn Haverkort. Mean-field analysis for the evaluation of gossip protocols. *2009 Sixth International Conference on the Quantitative Evaluation of Systems*, 2009.
- [4] Admin and Andras. Omnet++ discrete event simulator. <http://www.omnetpp.org/>.
- [5] Mathew Penrose. Lectures on random geometric graphs. *Random Graphs, Geometry and Asymptotic Structure*, page 67–101, 2016.
- [6] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, 2002.

# Appendix

```
import random

class Agent:
    def __init__(self, id, total_agents):
        self.id = id
        # Each agent starts knowing only their own message.
        self.known_messages = set([id])
        # Initially, every agent has a token.
        self.has_token = True
        # Keeps track of agents that this agent has communicated with.
        self.contacts = set()

    def communicate(self, other):
        """Exchange known messages with another agent."""
        self.known_messages |= other.known_messages
        other.known_messages |= self.known_messages

    def remove_token(self):
        """Remove the agent's token."""
        self.has_token = False
```

Figure 8.1: Class Agent

```
class GossipSimulator:
    def __init__(self, num_agents, max_rounds=10000):
        self.num_agents = num_agents
        self.max_rounds = max_rounds
        # Initialize the list of agents.
        self.agents = [Agent(i, num_agents) for i in range(num_agents)]
        # Count of total communications between agents.
        self.total_contacts = 0

    def get_agent_messages(self):
        """Retrieve the total messages known by each agent."""
        agent_messages = {}
        for agent in self.agents:
            agent_messages[agent.id] = len(agent.known_messages)
        return agent_messages
```

Figure 8.2: Class GossipSimulator

```
def _any_call(self):
    """ANY Protocol: Any agent can communicate with any other agent."""
    caller = random.choice(self.agents)
    callee = random.choice(self.agents)
    if caller != callee:
        caller.communicate(callee)
        self.total_contacts += 1
```

Figure 8.3: Anycall function

```

def _call_once(self):
    """CO Protocol: An agent communicates only once with another agent."""
    caller = random.choice(self.agents)
    # Get agents that haven't been contacted by the caller.
    eligible_callees = [a for a in self.agents if a.id not in caller.contacts and a != caller]
    if eligible_callees:
        callee = random.choice(eligible_callees)
        caller.communicate(callee)
        # Update contact sets for both agents.
        caller.contacts.add(callee.id)
        callee.contacts.add(caller.id)
        self.total_contacts += 1

```

Figure 8.4: Call Once function

```

def _spider(self):
    """SPI Protocol: Agents with tokens can call. The callee loses the token upon being contacted."""
    eligible_callers = [a for a in self.agents if a.has_token]
    if eligible_callers:
        caller = random.choice(eligible_callers)
        callee = random.choice(self.agents)
        if caller != callee:
            caller.communicate(callee)
            # Callee loses the token.
            callee.remove_token()
            self.total_contacts += 1

```

Figure 8.5: Spider function

```

def _learn_new_secrets(self):
    """LNS Protocol: An agent calls another agent only if they have at least one message the caller doesn't know."""
    caller = random.choice(self.agents)
    eligible_callees = [a for a in self.agents if a.known_messages - caller.known_messages and a != caller]
    if eligible_callees:
        callee = random.choice(eligible_callees)
        caller.communicate(callee)
        self.total_contacts += 1

```

Figure 8.6: LearnNewSecret function

```

def run_simulation(self, protocol):
    """Run the gossip simulation for the specified protocol until all agents know all messages or max rounds reached."""
    rounds = 0
    while any(len(a.known_messages) < self.num_agents for a in self.agents) and rounds < self.max_rounds:
        # Execute the chosen protocol.
        if protocol == "ANY":
            self._any_call()
        elif protocol == "CO":
            self._call_once()
        elif protocol == "SPI":
            self._spider()
        elif protocol == "LNS":
            self._learn_new_secrets()
        rounds += 1

    # Calculate performance metrics.
    avg_contacts = self.total_contacts / self.num_agents
    total_messages_known = sum(len(a.known_messages) for a in self.agents)

    # Reset agents and contacts for the next simulation.
    self.agents = [Agent(i, self.num_agents) for i in range(self.num_agents)]
    self.total_contacts = 0

```

Figure 8.7: RunSimulation function

```

import unittest
from dissertation import GossipSimulator, Agent

class TestGossipSimulator(unittest.TestCase):

    def setUp(self):
        self.simulator = GossipSimulator(num_agents=3)

    def test_init(self):
        self.assertEqual(self.simulator.num_agents, 3)
        self.assertEqual(self.simulator.max_rounds, 10000)
        self.assertEqual(len(self.simulator.agents), 3)
        self.assertEqual(self.simulator.total_contacts, 0)

```

Figure 8.8: Unit test

```

def test_get_agent_messages(self):
    agent_messages = self.simulator.get_agent_messages()
    self.assertEqual(agent_messages, {0: 1, 1: 1, 2: 1})

def test_any_call(self):
    self.simulator._any_call()
    self.assertEqual(self.simulator.total_contacts, 1)

def test_call_once(self):
    self.simulator._call_once()
    self.assertEqual(self.simulator.total_contacts, 1)

def test_spider(self):
    self.simulator._spider()
    self.assertEqual(self.simulator.total_contacts, 1)

```

Figure 8.9: Unit test 2

```

def test_learn_new_secrets(self):
    self.simulator.agents[1].known_messages.add(2)
    self.simulator._learn_new_secrets()
    self.assertEqual(self.simulator.total_contacts, 1)

def test_run_simulation(self):
    result = self.simulator.run_simulation("ANY")
    self.assertEqual(result["Protocol"], "ANY")
    self.assertTrue(result["Rounds taken"] > 0)
    self.assertTrue(result["Average contacts per agent"] > 0)
    self.assertTrue(result["Total messages known"] >= 3)

```

Figure 8.10: Unit test 3

```

class TestAgent(unittest.TestCase):

    def setUp(self):
        self.agent = Agent(0, 3)

    def test_init(self):
        self.assertEqual(self.agent.id, 0)
        self.assertEqual(self.agent.known_messages, {0})
        self.assertTrue(self.agent.has_token)
        self.assertEqual(self.agent.contacts, set())

    def test_communicate(self):
        other = Agent(1, 3)
        self.agent.communicate(other)
        self.assertEqual(self.agent.known_messages, {0, 1})
        self.assertEqual(other.known_messages, {0, 1})

    def test_remove_token(self):
        self.agent.remove_token()
        self.assertFalse(self.agent.has_token)

if __name__ == "__main__":
    unittest.main()

```

Figure 8.11: Unit test 4