UNIVERSITY OF ENGINEERING AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

WORKS PARTICIPATED
FACULTY LEVEL SCIENTIFIC RESEARCH STUDENT CONFERENCE

2023-2024

DHT-Based Cumulative Gossip Learning

Author: Le Anh Duc, 21020761, K66CD

Nguyen Tu Trung, 21020415, K66CD

Supervisor: Dr. Nguyen Dai Tho

HANOI - 2024

Abstract

Abstract: The growing complexity and scale of decentralized systems necessitate innovative approaches to information dissemination and learning. This research introduces a novel paradigm, "Distributed Hash Table (DHT)-Based Cumulative Gossip Learning," aimed at enhancing information sharing and cumulative learning in decentralized networks.

We propose a framework that leverages DHT to facilitate efficient and scalable communication among nodes. By integrating cumulative gossip learning mechanisms, the model fosters the exchange of knowledge and insights, allowing nodes to collaboratively accumulate expertise over time. This approach not only enables nodes to adapt to dynamic changes in the network but also enhances the overall learning capacity of the decentralized system.

Our study includes an in-depth exploration of the DHT architecture, the design of cumulative gossip learning algorithms, and their implementation in real-world decentralized environments. The results showcase the effectiveness of the proposed model in promoting information diffusion, knowledge consolidation, and improved decision-making within large-scale decentralized networks. This research contributes to the advancement of decentralized learning methodologies, offering valuable insights for the development of resilient and intelligent distributed systems.

Keywords: Decentralized collaborative learning, Gossip Learning, Peer-to-peer network, Distributed hash table.

THANKS

First and foremost, we would like to express our deep gratitude to all the professors at the University of Technology and Engineering. In particular, we extend our heartfelt thanks to our mentor, Dr. Nguyen Dai Tho, who has guided us, provided invaluable insights, and dedicated as much time as possible to assist us in shaping the direction of this research. Moreover, we sincerely appreciate the support and encouragement from our classmates during challenging times.

We commit that this is our own work, and it is not copied from anyone. In the event of any violation, I willingly accept all consequences and responsibilities as per the regulations of the University of Engineering and Technology

Sign of supervisor

Contents

List of Abbreviations

CFL	Centralized Federated Learning
DFL	Decentralized Federated Learning
DHT	Distributed Hash Table
DL	Deep Learning
DNN	Deep Neural Networks
DSGD	Decentralized Stochastic Gradient Decent
FL	Federated Learning
GL	Gossip Learning
ML	Machine Learning

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Today, rather than opting for centralized learning, modern research has shifted its focus towards the Federated Learning model. This method allows model training on distributed data without the need to centralize the data. Thanks to its decentralized nature, Federated Learning helps minimize privacy risks and enhances machine learning performance in complex environments.

Federated Learning is an advanced machine learning approach, yet it faces notable drawbacks. One of the major challenges is the model retraining time, as the process involves collecting and combining information from numerous distributed devices, resulting in significant time consumption. When a large number of devices participate, this can impact the swift deployment of model updates and introduce delays in real-world applications.

Another drawback is the substantial amount of exchanged messages among participating devices. When devices have to send model update information to a central hub or exchange information with each other, it demands considerable bandwidth and network energy. For systems with a high number of devices or frequent model updates, the exchange of messages can become a bottleneck, affecting network performance and resource consumption.

In the broader field of computer networks, there have been solutions utilizing distributed hash tables (DHTs) for effective information aggregation. About the applications of Distributed Hash Tables (DHT) to cyber security. Min Cai $et\ al$ [2] proposed collaborative worm containment over distributed hash table (DHT). Kien $et\ al$ [3] pro-

posed collaborative intrusion detection in terminal devices using a distributed hash table (DHT). However, no DHT method has been applied to gossip learning, except for a work by Dung $et\ al$ [4]. As this is the first paper addressing this issue, it has some limitations, such as a high number of sent messages, long convergence time for ratios, and, most notably, a complexity returning to $O(N^2)$ after a certain period.

In our work, we propose a method applying DHT to gossip learning more effectively. Regarding the number of messages, we send fewer, and the convergence time for ratios is faster, with no compromise in accuracy compared to the previous method.

The structure of this scientific research is as follows: Chapter 2 provides an state of the art include overview of federated learning and gossip learning. In Chapter 3, we detail our experimental setup, model, the parameters selection and other information.Next, In chapter 4, we delve into our experimental results. Finally, Section 5 serves as the conclusion to the scientific research.

Chapter 2

State of the art

2.1 Overview of federated learning

Federated Learning represents a significant breakthrough in the field of machine learning, where the model training process doesn't require transferring the entire dataset to a centralized location. Instead, it is performed directly on distributed devices. This approach addresses significant challenges, including privacy concerns and network bandwidth pressure.

The process begins by initializing a global model, which is then distributed to participating devices. Each device then trains its model based on local data. Instead of transmitting the entire model or dataset, only update information, such as new model parameters, is sent to the central hub.

The central hub then aggregates update information from all devices and updates the global model. This process repeats until the model achieves the desired performance.

Federated Learning offers numerous benefits, with privacy protection being paramount. By keeping data in place and sharing only necessary information, it mitigates risks associated with personal information leakage. Simultaneously, by reducing network bandwidth pressure through the avoidance of large data transfers, Federated Learning becomes an efficient choice for environments with numerous distributed devices.

2.1.1 Categories of Federated Learning

Figure 2.1 show the illustration of different learning approaches includes local learning, centralized learning, Centralized Federated Learning (CFL), and Decentralized Feder-

ated Learning (DFL). In the local learning scenario 2.1a, individual clients are trained exclusively on their respective user data without sharing raw data or communicating with one another. In centralized learning 2.1b, clients transmit user data packets to a server, which then trains a global model using the entire dataset; this generalized model is subsequently shared with all clients. For Centralized Federated Learning (CFL) 2.1c, clients send their locally trained model parameters to the server, which aggregates these local models and transmits the consolidated global model parameters back to all clients. Lastly, in Decentralized Federated Learning (DFL) 2.1d, clients exchange their locally trained models, enabling subsequent clients to learn, personalize, and adapt the model locally while also sharing and propagating model parameters that encapsulate local knowledge.

2.1.2 Federated Learning

Federated Learning is a deep learning category that enables training models across decentralized devices or servers while keeping data localized. In traditional machine learning models, data is typically collected and centralized on a server for training. The fusion of contemporary deep learning algorithms with vast datasets renders deep learning technology a promising solution for addressing complex real-world challenges. Nonetheless, traditional centralized approaches are fraught with numerous issues.

Firstly, in practical terms, the majority of enterprises, excluding a few large corporations, possess limited and often insufficient data quality to sustain the deployment of data-intensive AI services.

Secondly, accessing raw data from participants poses privacy concerns. Particularly for enterprises, data obtained from commercial sources holds significant potential value. However, companies, even different departments within the same organization, are often hesitant to share data. On an individual level, most personal data includes sensitive information such as travel history, health status, financial data, and more.

Thirdly, centralized training methods require substantial computational resources, resulting in unacceptable delays. Conversely, the challenges posed by limited device resources and an unstable network environment complicate distributed machine learning. In response to these challenges, Federated Learning, a distributed learning framework that preserves the privacy of training data, has garnered growing attention. This approach enhances model effectiveness by aggregating models from multiple clients while ensuring security and privacy.

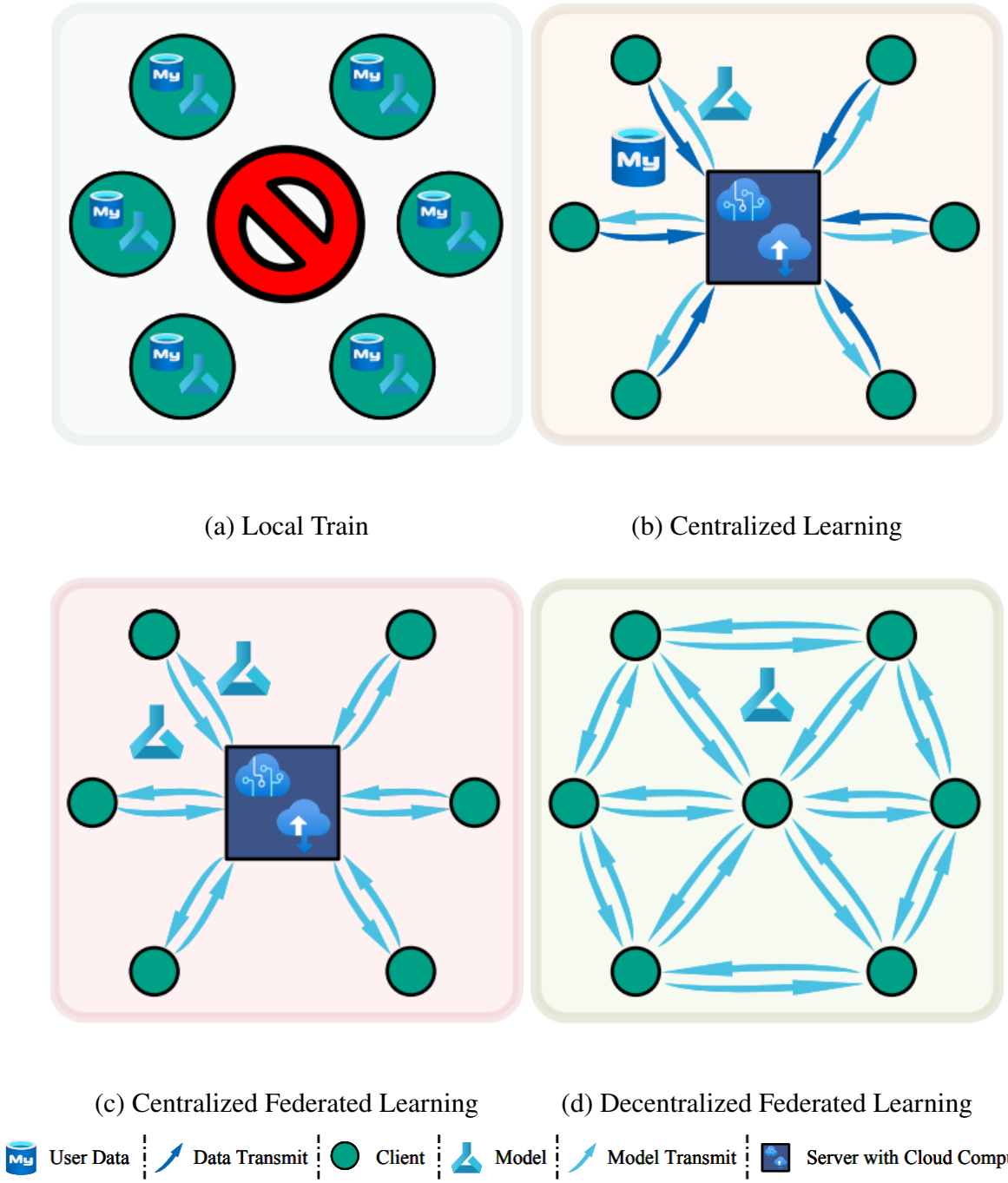


Figure 2.1: The categories of Federated Learning

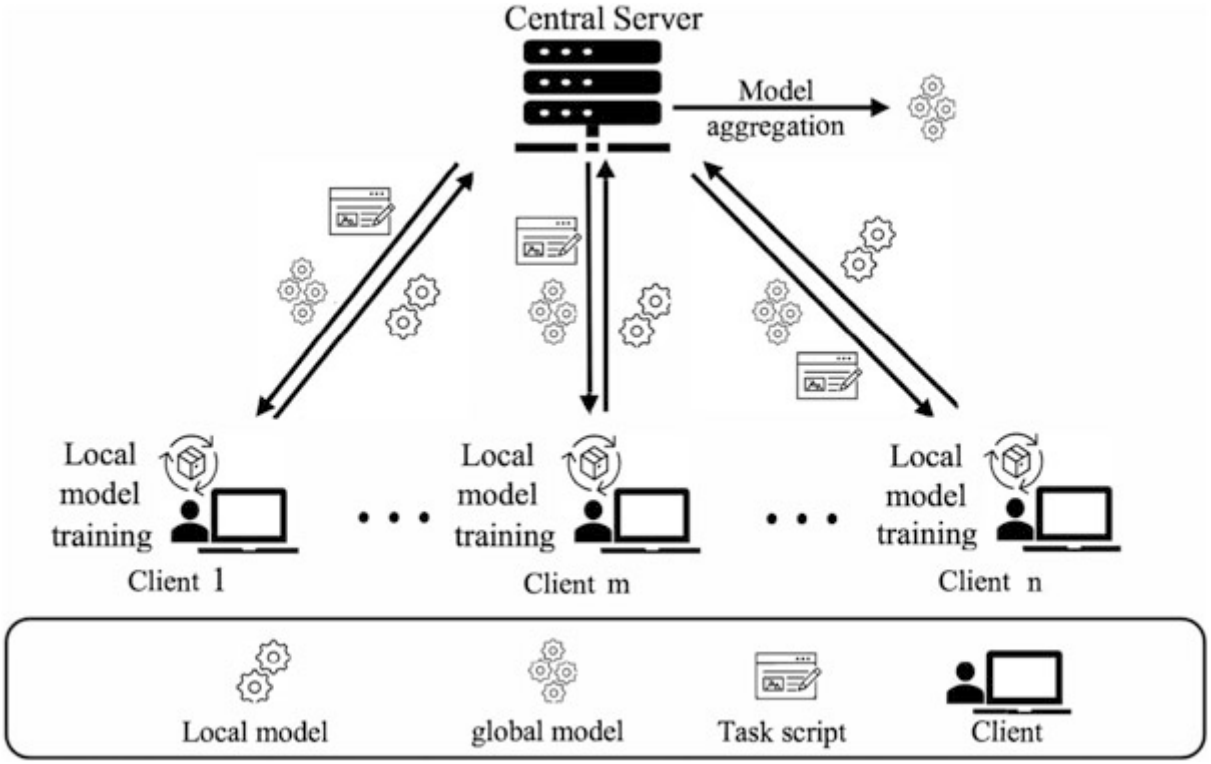


Figure 2.2: A typical Federated Learning framework

In the figure 2.2, we depict a standard Federated Learning process comprising the following four steps:

1. The server dispatches a shared model to selected clients.
2. Participating clients refine the model with their local data and send the updated model or parameters back to the server.
3. The server aggregates the updates and broadcasts the combined model to all clients.
4. The aforementioned steps iterate until convergence to a global model or a predefined number of rounds is reached.

The pseudo-algorithm 1 representing FederatedAveraging released with Federated Learning in 2015 by Google researchers [8]. The distinctive innovation of Federated Learning lies in the server's need for parameters only, eliminating the necessity to collect raw data from clients. This design safeguards sensitive information, as each client retains its data locally. This unique capability has led to widespread adoption in both industry and academia, offering privacy protection in diverse modern applications such as mobile keyboard prediction, personalized medicine, traffic forecasting, anomaly detection, and more.

Function `Server_executes()`**:**

Initialize ω_0

for *each round* t *= 1, 2, ...* **do**

$m \leftarrow max(C \cdot K, 1)$

$S_t \leftarrow$ random set of m clients

for *each client* $k \in S_t$ **do**

$w_{t+1}^k \leftarrow ClientUpdate(k, w_t)$

end

$w_{t+1} \leftarrow \Sigma_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

end

end

Function `ClientUpdate()`**:** // *Run on clients*

for *each local epoch* i *from* $1 \rightarrow E$ **do**

$\omega \leftarrow \omega - \eta \nabla l(\omega;\ b)$

end

return ω to server

end

Algorithm 1: FederatedAveraging presented in 2015 by Google researchers

2.1.3 Challenge of Federated Learning

Numerous applications have embraced Federated Learning, allowing users to train models on their mobile devices without uploading raw data. However, this technique confronts critical challenges. The decentralized nature of distributed learning, coupled with localized data storage, renders Federated Learning susceptible to attacks by malicious nodes. Additionally, challenges arise from the heterogeneity of users and datasets, compounding processing difficulties. Beyond the general challenges of deep learning, Federated Learning introduces unprecedented issues and is vulnerable to threats impacting classical learning algorithms. Outlined below are four primary challenges in Federated Learning systems from our perspective:

1. Security: Security challenges encompass the absence of data access control and attacks from malicious nodes, typically launched by hackers targeting the system or model. Poisoning attacks, such as fake label attacks and backdoor attacks, can cause severe damage if the number of poisoned datasets is substantial.

2. Privacy: While Federated Learning protects privacy by sharing local model pa-

rameters instead of clients' raw data, merely maintaining data localization during training does not guarantee sufficient privacy. Analyzing gradient information can still reveal private knowledge to third parties or central servers, posing risks like image recovery and launching inversion and inference attacks.

3. Communication Cost: In Federated Learning systems, involving millions of devices, each device may spend significantly less time training models locally than engaging in network communication. The periodic uploading of local models to the server intensifies communication costs, especially with a large number of participants. Limited wireless network bandwidth further exacerbates the issue, and optimizing uplink communication with asymmetric connection speeds presents a significant challenge.

4. Heterogeneity: The diverse network status, storage, and processing capabilities of devices introduce heterogeneity, impacting computing and communication capabilities. This diversity complicates delayed mitigation and fault tolerance. Furthermore, variations in data generation and collection methods contribute to data heterogeneity, posing challenges in modeling and evaluating non-IID (non-identically distributed) data.

2.1.4 Variants of Centralized Federated Learning

The network variations and expansions of CFL have been devised to tackle the challenges mentioned above and adapt to diverse real-world application scenarios.

1. Edge Variant: In edge networks, FL typically undergoes additional aggregations through the setup of extra edge servers. These servers aim to distribute communication and computing pressure, mitigating the impact of a Single Point of Failure (SPoF). Being geographically closer to clients, these edge servers result in reduced communication resource consumption and lower latency. Apart from communication optimizations, the geographic proximity of edge servers to clients may lead to better adaptation to the connected clients. Edge servers and connected clients can be viewed as geographically personalized clusters. Blockchain plays a crucial role in the edge variant, creating a robust security barrier, managing resources, and ensuring reliable communication in the network. FL with blockchain aggregates the global model through a distributed shared ledger, resulting in reduced burden on the central server, enhanced security, confidence, and decreased communication

costs. Nguyen et al [9] introduced FLchain to fulfill the requirements for servers and communication resources in the blockchain through mobile edge computing servers.

2. Personalization Variant: Personalized FL can be categorized into global model personalization and personalized model architecture. Global model personalization typically begins with a global model, which the client then personalizes to fit local user preferences. Personalization on the client side, independent of the server, includes techniques like federated transfer learning to transfer global model knowledge locally. Personalized model architecture modifies the traditional FL architecture to create a personalized model with user knowledge, driven by server behavior. Clustered FL [10] is a well-known architecture in this regard. The client model in the personalized FL framework is closer to the user, known for its high accuracy and confidence. Particularly effective for non-independent and non-identically distributed (non-IID) data, personalization can transfer and adapt the model to different heterogeneities when the aggregated global model deviates from the user.

3. Split Variant: Split FL involves splitting the model for learning, where the server handles certain model layers. The client only sends the hidden representations and/or gradients in the cut layer of the model to the server. This framework not only shifts part of the learning task to the server but also avoids sharing user data. Compared to traditional FL, the split FL framework [**?**] maintains similar accuracy and communication efficiency with a reduced learning burden on the client side and enhanced privacy protection. However, split FL is still in its early stages and faces significant limitations, such as the need for more communication resources, particularly with the potential presence of a Single Point of Failure (SPoF) on the server.

Variations of CFL currently exist with diverse frameworks, including single, multiple, sub, and master servers to optimize and target different problems. In addition to these variants, a popular approach involves assembling various FL framework variants to address multiple issues.

2.1.5 Decentralized Federated Learning

2.1.5.1 Decentralized Federated Learning taxonomy

In this segment, we will begin with the analysis and comparison of DFL alongside related designations. The difference in types of DFL is indicated by the diverse iterations, protocols, network topologies, and variations within DFL, as outlined below. Moreover, these taxonomies can be employed in a complementary fashion. They represent the perspectives of the encapsulating summarizations of existing literature. This all-encompassing approach is designed to enhance the comprehension and categorization of concepts within the domain of DFL.

- A. Ordering of Iterations: Generally, Federated Learning (FL) necessitates multiple iterations for convergence, and the iteration order denotes the sequence of each client in every iteration or the formation of client queues in Distributed Federated Learning (DFL). In Centralized Federated Learning (CFL), clients iterate concurrently, and the order in which the server receives client models does not influence system convergence. However, in DFL, the iteration order of clients significantly impacts the performance of client models, a topic thoroughly examined in Section III-D. Depending on the specific usage scenario and task requirements, the client iteration order in DFL can be established as sequential, cyclic, random, parallel, dynamic, or other strategies. The selection of the iteration order holds sway over the convergence and performance of the system, underscoring the importance of accounting for the specific characteristics and constraints of the application when determining the suitable order.

- B. Network Communication Protocol: Distributed Federated Learning (DFL) serves as a network framework facilitating the exchange of model weights through pointing, gossip, or broadcast protocols, aiming to achieve optimal models across all clients. Pointing represents a straightforward unidirectional, one-to-one, and specified communication relationship between two peers. Gossip and broadcast algorithms, well-established in networks, offer stochastic communication methods. The gossip protocol entails a random peer-to-peer approach for clients to share and disseminate knowledge, serving as a standard communication protocol in DFL. In its early stages , the broadcast protocol adopts a one-peer-to-all-peers approach, enabling a client to broadcast its model to all others.

Hybrid protocols are gaining popularity, presenting varied combinations of gossip, broadcast, and their integrated communication structures tailored to diverse scenarios and constraints.

- C. Network Topology: The architecture of Distributed Federated Learning (DFL) networks draws inspiration from network topology. The diversity of DFL networks is evident in figure 2.3, attributed to the absence of server adaptation, management, and propagation constraints. It is important to note that the line segment merely signifies a connection between clients, which can be either unidirectional or bidirectional. Additionally, the transmitted content need not be limited to the client's model; it may also encompass the model of a previous client. The computational content of a client extends to include local learning and aggregation. Considering the pivotal role that communication protocols play in Distributed Federated Learning (DFL), a crucial question arises regarding how to enhance the efficiency of knowledge dissemination. Using figure 2.3 as an illustration, we outline the sequential pointing line DFL process:
  - Client 1 learns from the initial model based on its local knowledge.
  - Client 1 sends the locally trained model (Model 1) to Client 2.
  - Client 2 engages in continual learning on Model 1, resulting in Model 2.
  - Client 2 transmits two types of content to Client 3:
    - a. Only Model 2.
    - b. Including both Model 1 and Model 2.
  - Client 3 has two ways to obtain Model 3:
    - a. Client 3 uses Model 2 for continual learning to obtain Model 3.
    - b. Client 3 aggregates Model 1 and Model 2 to obtain Model 3', then uses Model 3' for local learning to obtain Model 3.

  Repeat this process until the last client.

  While the sequential pointing line form of DFL is established, various considerations and variations exist, including different transmission and aggregation options. In Centralized Federated Learning (CFL), aggregation proves to be the fastest and most efficient way to integrate all client knowledge. However, for DFL, challenges

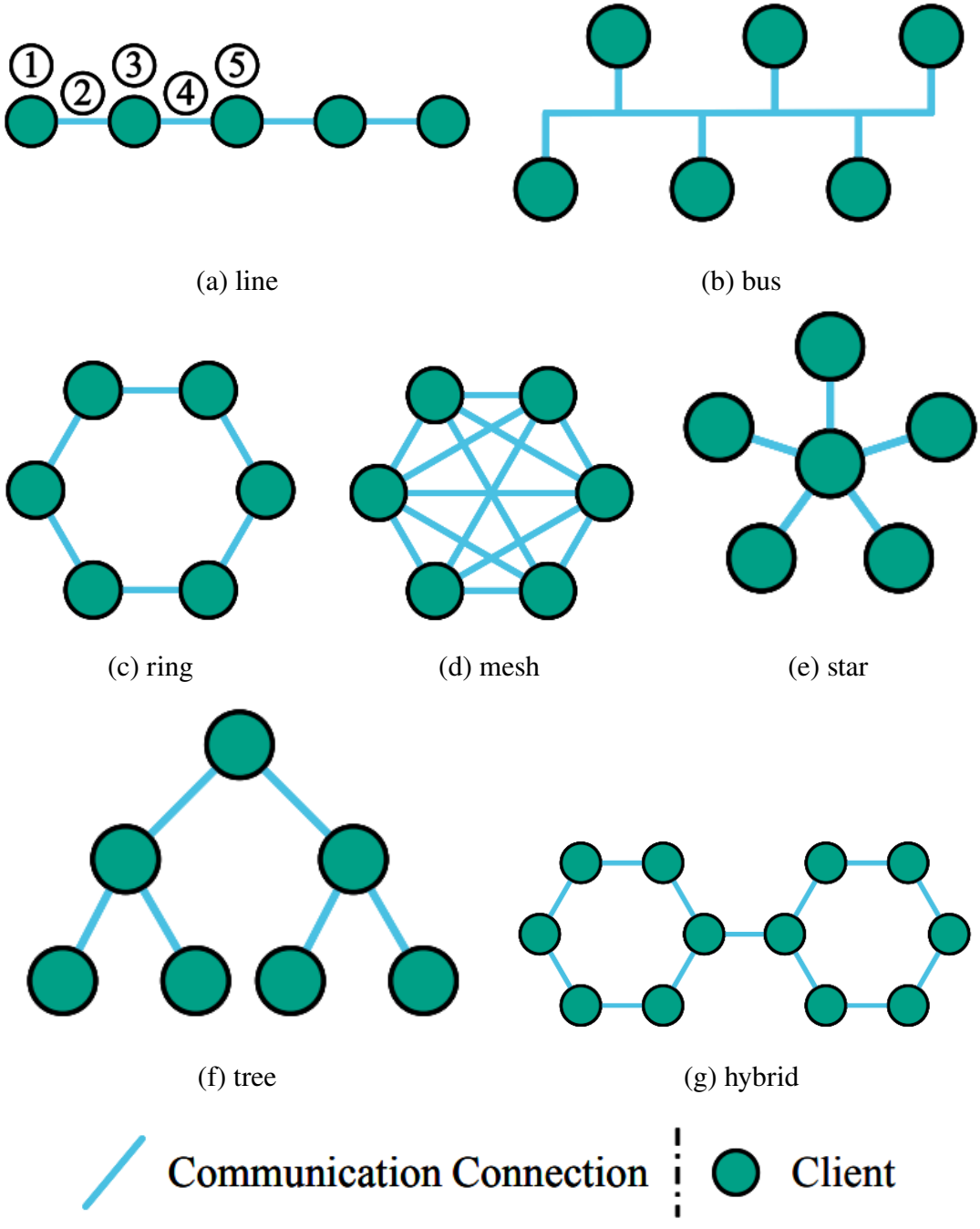
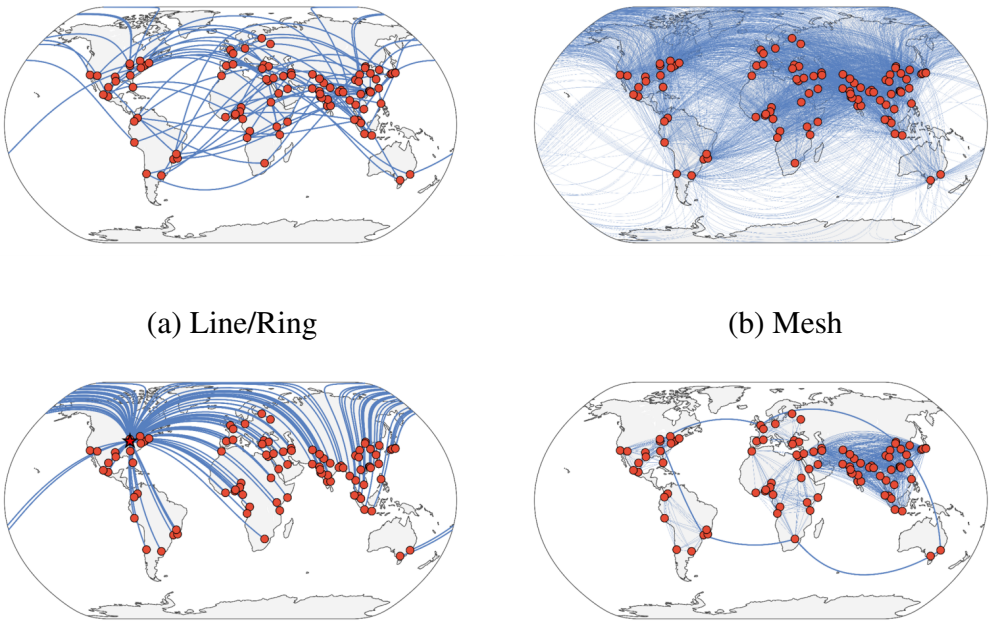


Figure 2.3: The topologies of Decentralized Federated Learning (DFL)



(a) Line/Ring (b) Mesh

(c) Star (d) Hybrid

Figure 2.4: The topologies of Decentralized Federated Learning (DFL) in real world

emerge due to diverse network topologies, different model versions, and the complexity of acquiring all client knowledge without a centralized server for collaborative management. Urgently needed is an alternative paradigm that complements and expands the Federated Learning landscape, especially in scenarios where aggregation is not well compatible.

2.2 Gossip Learning

Gossip Learning is a category of the Decentralized Federated Learning (DFL) with the original communnication way is broadcast-gossip. In detail, the clients communicate in a random form of one-peer-to-other-peers, which varies among the implementation.

Now we describe our variant of Gossip Learning in this scientific research through the remaining taxonomies reprented above. The iteration order of our proposal is in parallel when all clients communicate asynchronously. The network we use for all-to-all protocol topology is mesh (fully connected topology) which is depicted by figure 2.3d and 2.4b. While in our DHT-based protocol, the topology is not a standard one, it can be understood as an extended ring topology. The paradigm proposal is $aggregate$ type where each node in the network receives some models per iteration then it aggregates each time they receive it and then learns on the aggregated models. The learning activity

mentioned happens in an asynchronous cycle of each node. Finally, the communication architecture will be static, we only initialize it once the learning process starts, then it remains unchanged through the end of the process. Algorithm 2 will show how Gossip Learning logically operates.

Function `start()`**:**
    Initialize the weight (ω_k), bias(b_k), and iteration (t_k)
    while *true* **do**
        wait(Δ)
        $p \leftarrow selectPeer()$
        send sample(t_k, w_k, b_k) to p
    end
    Function `onReceiveModel()`**:**
        mergeModel($(\omega_k, b_k, t_k), (\omega_r, b_r, t_r)$)
        updateModel($(\omega_k, b_k, t_k), D_k$)
    end
end

Algorithm 2: Gossip Learning algorithm

2.2.1 All-to-all protocol

Here, we represent the all-to-all protocol. In each iteration, the nodes send their model to each other which is shown in Algorithm 3. After receiving a model, they merge and update their model to finish the iteration. All-to-all protocol seems more like a broadcast version of Decentralized Federated Learning when the nodes will communicate with all of the other nodes for their model update. By applying this to Gossip Learning, we have an elementary approach for our proposal comparison.

Input : The ith node and its model
Output : $\emptyset$
Function `send(`i, θ`)`**:**
    $node \leftarrow nodes[i]$
    $model \leftarrow \theta$
    for $peer$ ***in*** $node$***.get_peers()*** **do**
        $node$.send($peer$, $model$)
    end
end

Algorithm 3: Model sending in All-to-all protocol

Moreover, both all-to-all protocol and our proposal use decentralized SGD as their optimizer for the experiment reprented later.

2.2.2 Decentralized stochastic gradient descent

Next, we introduce the Decentralized stochastic gradient descent (DSGD). Recently, there has been significant interest in decentralized stochastic optimization methods, primarily due to their advantages:

- Scalability: Decentralized SGD allows for parallelization of model training across multiple nodes or devices. This enables efficient use of computational resources and accelerates the training process, making it suitable for large datasets and complex models.
- Low latency: By performing local computations on each node, decentralized SGD can reduce the latency associated with sending data to a central server. This is particularly beneficial in scenarios where low-latency updates are critical, such as in online learning or real-time applications.
- Fault Tolerance: Decentralized architectures are often more robust to node failures. If a node in the decentralized network fails or drops out, the training process can continue with the remaining nodes. This enhances the overall fault tolerance of the system.
- Communication efficiency with asynchronous updates: Decentralized algorithms can allow for asynchronous updates, where nodes communicate and update their models independently. This asynchrony can lead to improved communication efficiency, as nodes are not required to wait for the completion of updates from all other nodes before proceeding.
- Global model consistency: Decentralized training algorithms are designed to ensure that all nodes converge to a consistent global model. This is achieved through communication and aggregation strategies that maintain model coherence across the decentralized network.
- Distributed learning in edge computing: Decentralized SGD is well-suited for edge computing environments where training occurs on distributed edge devices. This is essential for applications where data is generated and processed at the edge, reducing the need for centralized data transfer and processing.

There are many customized versions in the family of decentralized SGD variants; nevertheless; we use only the standard one which should be called gossip averaging using the fixed mixing matrix. We do not require a unique variant of decentralized SGD because this is not a special case, just a standard one in Gossip Learning.

Chapter 3

Our proposition

3.1 Overview

In Nguyen Ba Nam Dung's thesis, he proposed using distributed hash tables for network topology. We will now describe the basics of the method. The method with recurrence order is parallel when all clients communicate asynchronously. The network used for the all-to-all protocol mesh topology (fully connected topology) is depicted in Figures ... and... Although in the DHT-based protocol of the method, the topology is not a standard structure but it can understood as an extended ring structure. The method is a type of aggregation where each node in the network receives some model on each iteration, then re-aggregates each time they receive it and then learns according to the aggregate model. The mentioned learning operation occurs in asynchronous cycles of each node. Finally, the communication architecture will be static, the method will initiate the learning process to begin with then will not change until the end of the process. But after research and experimentation, we realized that this method still has many imperfections. We would like to present some of the criteria that have emerged:

1. Although this method reduces the number of neighbor nodes from $O(N^2)$ to $O(Nlog(N))$, if run in a long loop, the method will become $O(N^2)$ as before.

2. The convergence point when sending messages is still slow, the method needed to send about 120000 messages to get the convergence point is corresponding to f1-score of 0.8.

3. The number of messages sent is still high, leading to increased time.

3.2 Proposed method

Building upon the shortcomings identified in our preceding methodology, we embarked on a thorough reevaluation, ushering in a paradigm shift in our approach. The cornerstone of our novel strategy lies in the strategic application of Distributed Hash Table (DHT), intricately weaving a tree structure that meticulously eliminates any semblance of node duplication.

In the initial stage, emanating from a carefully chosen parent node, an expansive dissemination of information pertaining to the first round permeates through the entire network, reaching each leaf node with precision. Subsequently, the leaf nodes engage in a sophisticated process of synthesizing, progressively accumulating, and then meticulously transmitting the refined information back to the root node.

At the apex of this intricate process, the root node, armed with the amalgamated insights from the network's periphery, exercises its decision-making prowess to arrive at a conclusive determination. The distinctive advantage of this refined methodology lies in its exceptional capacity to drastically reduce the volume of transmitted messages, resulting in a computational complexity of O(N). Moreover, this approach exhibits an unparalleled ability to achieve both rapid and highly stable convergence rates, establishing itself as a beacon of efficiency in distributed computing paradigms.

3.3 Proposed algorithm

To enhance the criteria mentioned above, our collaborative research team has further refined the method presented in the thesis. Initially, we made a minor adjustment to the node initialization process. The distributed hash table needs to be incorporated into the individual node creation process as a list of its neighboring nodes. This is a significant improvement to our methodology.

Firstly, we try to do a little change in the node initialization, the distributed hash table needs to be a part of the creation of an individual node as a list of its list of neighbors. See the Algorithm 4 for the details.

3.3.1 Node initialization

In the Algorithm 4, we reverse the order of the table as a little modification for the simplicity of other processes.

Algorithm 4: Initialize a node in DHT-based protocol

Input : N - number of nodes

Function `initializeNode(`N`)`**:**

```
bits ← log2N
finger ← an empty array
for i ← 0 to bits do
    peer ← successor(idx + 2^i)
    finger[i] ← peer
end
finger.reverse()
```

end

3.3.2 Tree Initialization

Next we will initialize a tree for data transfer. The input is a node we choose as the root node, the child nodes are the neighbors created in algorithm 4 but with the condition that the nodes can only appear once, for example node 1 is a neighbor of 0 and 99. And when creating a tree, node 1 is a child of node 0, then node 99 will no longer have a child of node 1.

3.3.3 Broadcast Initialization

After the tree creation step, the root node will send messages to the child nodes that have been created and stored in a hash table. The child nodes continue to forward messages to the next child nodes (during this process, the data has not been learned). and update. It is noteworthy that this figure is only served for the illustration of the algorithm, in the experiment of our proposal, N the number of nodes is 100 instead.

3.3.4 Broadcast Return

After the root node transmits the message to all child nodes, to the last node, the process will be reversed, the parent node will pull data from the child nodes to update data and weights, continue. Thus, reaching the root node ends the process. 7.

Algorithm 5: Tree Initialization

Input : Int index idx

Output : Dictionary of Chord Nodes

Function `buildChordTree(`idx`)`**:**

$usedChord \leftarrow$ set() $queue \leftarrow [idx]$

while *queue not empty* **do**

$i \leftarrow$ remove first from queue add i to $usedChord$

for *each* $fingerIdx$ *in* $nodes[i].finger$ **do**

if $fingerIdx$ *not in* $usedChord$ **then**

append $fingerIdx$ to $queue$ add $fingerIdx$ to $usedChord$

end

else

remove $fingerIdx$ from $nodes[i].finger$

end

end

end

return *nodes*

end

Algorithm 6: Initialize Broadcast in DHT-based Protocol

Input : The node i and its model θ

Function `initBroadcast(`i, θ`)`**:**

self.nodes = self.build_chord_tree(i) $queue \leftarrow [i]$

while *queue not empty* **do**

$node\_idx \leftarrow$ remove first from queue **if**

self.nodes[$node\_idx$*].timed_out(t, W_matrix[*$node\_idx$*])* **then**

for *each* $finger\_idx$ *in self.nodes[*$node\_idx$*].finger* **do**

$msg \leftarrow$ self.nodes[$node\_idx$].send(t, $node\_idx$, $finger\_idx$, self.protocol)

end

end

end

end

Algorithm 7: Return Broadcast in DHT-based Protocol

Input : The node i, model θ, time t, message queues msg_queues, reply queues rep_queues

Function `broadcastReturn` (i, θ, t, msg_queues, rep_queues) **:**

- $is\_online \leftarrow$ random$(self.n_nodes) \leq self.online_prob$
- **for** *each* msg *in* $msg\_queues[t]$ **do**
  - **if** $is\_online[msg.receiver]$ **then**
    - $reply \leftarrow self.nodes[msg.receiver].receive(t, msg)$ **if** $reply$ **then**
      - **if** *random*$() > self.drop\_prob$ **then**
        - $d \leftarrow self.delay.get(reply)$ append $reply$ to $rep\_queues[t + d]$
      - **end**
    - **end**
  - **end**
- **end**
- **for** *each* $reply$ *in* $rep\_queues[t]$ **do**
  - **if** $is\_online[reply.receiver]$ **then**
    - self.notify_message(False, reply)
    - self.nodes[reply.receiver].receive(t, reply)
  - **end**
- **end**

end

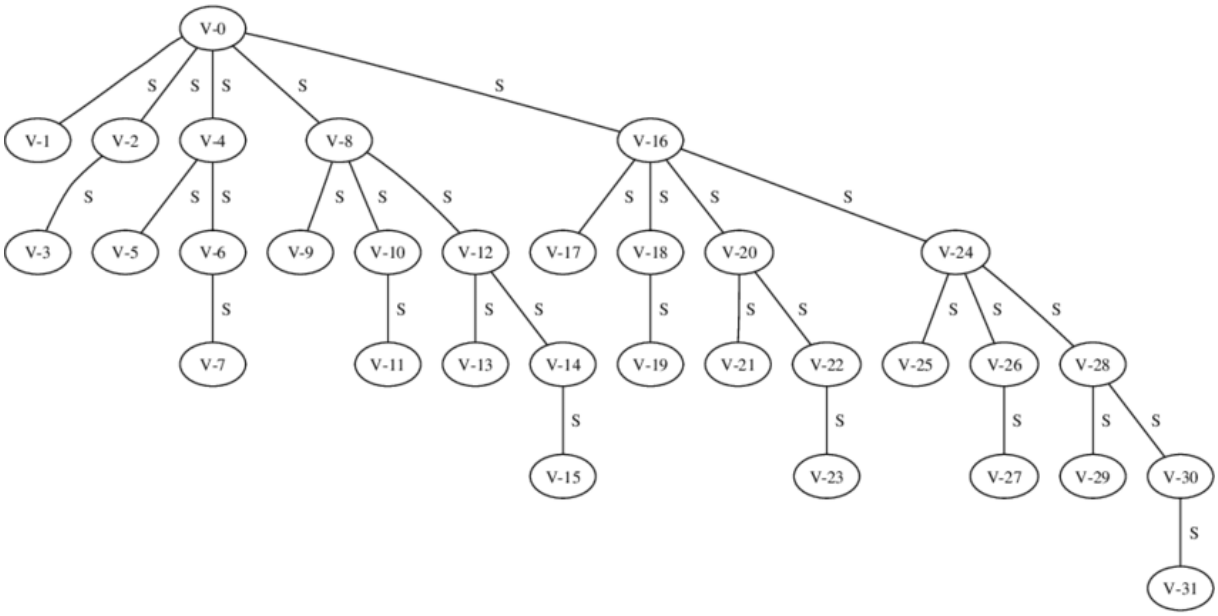


Figure 3.1: The broadcast tree of the DHT-based protocol

Chapter 4

Experiments and results

4.1 Experimental Configuration

4.1.1 Dataset

We used the Spambase[5] dataset (SPAM E-mail Database) to test the performance of the All-to-all and DHT-based protocol. This dataset contains a collection of emails which we have to classify into two groups of labels showing that an email is spam (ham) or not. About the data, each email has been pre-processed and transformed into a set of attributes or features. These features typically include characteristics such as the frequency of certain words, the presence of specific terms, and other relevant metrics extracted from the email content.

For more detail, the Spambase dataset is derived from the publicly available corpus provided by the Apache SpamAssassin project. It is a tabular format, with rows representing individual emails and columns representing features. The dataset contains a moderate number of instances, typically around 4,601 rows. For Spambase, a common distribution is around 60% non-spam (0) and 40% spam (1). This balance helps ensure that a machine learning model trained on this dataset is exposed to a representative amount of both classes, enabling it to learn patterns associated with both spam and non-spam emails.

The dataset has been widely used in academic and industry research to develop and benchmark various machine learning models for binary classification. The following table 4.1 shows some variables representing an email.

Table 4.1: Variables table of Spambase dataset

Variable Name	Role	Type	Description	Missing value
word_freq_cs	$Feature$	$Continuous$		no
word_freq_business	$Feature$	$Continuous$		no
word_freq_address	$Feature$	$Continuous$		no
word_freq_credit	$Feature$	$Continuous$		no
word_freq_technology	$Feature$	$Continuous$		no
char_freq_;	$Feature$	$Continuous$		no
char_freq_[	$Feature$	$Continuous$		no
char_freq_$	$Feature$	$Continuous$		no
capital_run_length_average	$Feature$	$Continuous$		no
capital_run_length_longest	$Feature$	$Continuous$		no
capital_run_length_total	$Feature$	$Continuous$		no
Class	$Target$	$Binary$	spam (1)/not spam (0)	no

4.1.2 Tools

In this thesis, we use a device with configuration:

- $1 \times$ AMD RYZEN 7 5000 SERIES
- $1 \times$ AMD RADEON RX 5500M
- $1 \times$ 8GB DDR4 @ 3200MHz
- $1 \times$ 8GB DDR4 @ 3200MHz

We employ Python 3.11 and PyTorch for our software development.

4.1.3 Model and optimizer

Our machine learning model in each node is logistic regression. Logistic regression 4.1 is a statistical method used for binary classification, where the goal is to predict the probability of an observation belonging to a particular category. Despite its name, logistic regression is employed for classification rather than regression tasks. It models the relationship between a dependent binary variable and one or more independent variables by employing the logistic function. This function transforms a linear combination

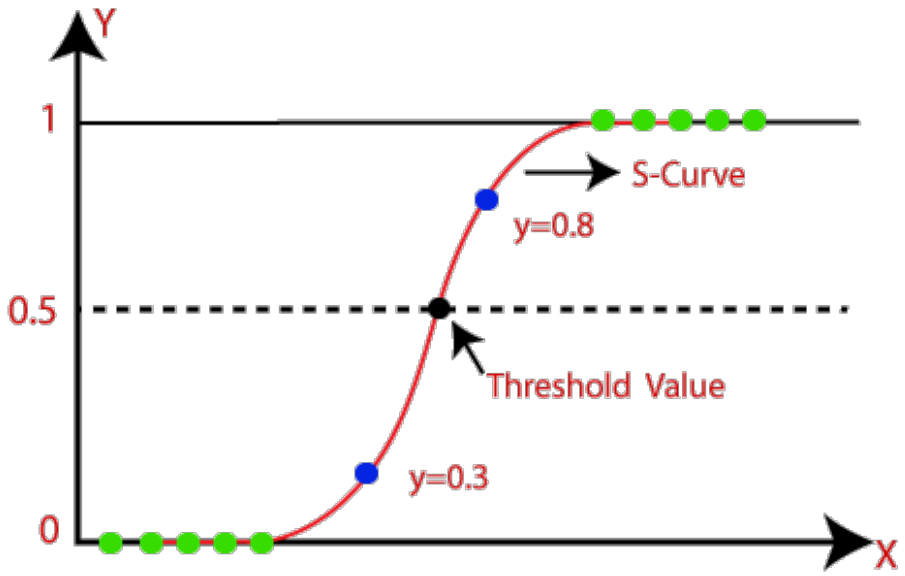


Figure 4.1: Logisic regression

of input features into values between 0 and 1, representing the probability of the event occurring. The model is trained by adjusting the parameters through a process called maximum likelihood estimation, which seeks to maximize the likelihood of observing the given outcomes.

The optimizer in this study is decentralized stochastic gradient descent (Decentralized SGD) which have drawn a large amount of attention [7], [6] and [1] and likely become the best optimizer for almost all the distributed computing environment lately because thanks to their cost per iteration, data locality and their communication efficiency. Therefore, we choose this optimizer for the experiment as a popular approach.

4.1.4 Parameters

About some experiments, we use the hyper-parameters for the decentralized SGD and other basic parameters the deep learning model in table 4.2

About the number of rounds, we simulate the Gossip Learning network and estimate the suitable round for the model to converge in the appropriate time. In addition, only a part of the Spam-base dataset is enough for training so the test_size is set to 0.1 (10%) of the total samples.

Similar to other approaches, we set the online proportion of the nodes to 100% the drop proportion of the message is 0%.

Table 4.2: The parameters used in the experiment

Parameter	Value
number_of_node	100
learning_rate	0.1
weight_decay	0.01
round_length	100
test_size	0.1
drop_prop	0
online_prop	1

4.1.5 Evaluation metric

Firstly, we try various metric values similar to other evaluations of the studies : 0-1 Error, precistion, recall, F1 score, the number of labels in the test set that the model misclassifies. This value is calculated by the elements in the confusion matrix 4.3 which allows the visualization and assessment of the performance of a classification algorithm on a set of data for which the true values are known. The matrix consists of four elements that represent different combinations of predicted and actual class labels.

$True\ positive(TP)$ = Number of samples whose actual class is positive, and the model correctly identifies it as positive

$False\ positive(FP)$ = Number of samples whose actual class is negative, but the model incorrectly identifies it as positive. False positives are also known as Type I errors

$True\ negative(TN)$ = Number of samples whose actual class is negative, and the model correctly identifies it as negative

$False\ negative(FN)$ = Number of samples whose actual class is positive, but the model incorrectly identifies it as negative. False negatives are also known as Type II errors

$$Accuracy = \frac{Num(TP) + Num(FN)}{Num(TP) + Num(FN) + Num(TN) + Num(FP)}$$

$$0 - 1\ Error = 1 - Accuracy$$

Table 4.3: Confusion Matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

$$Precision = \frac{Num(TP)}{Num(TP) + Num(FP)}$$

$$Recall = \frac{Num(TP)}{Num(TP) + Num(FN)}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

4.2 Results

4.2.1 Convergence speed

The results show that all three algorithms converge equally, but the new hash table algorithm converges faster due to the preparation of nodes before sending. In Figure 4.5, to achieve an F1 score of 0.86, the DHT and all-to-all algorithms need to send about 60,000 messages, while the new DHT algorithm only needs to send 12,000 messages. This shows that the algorithm is achieving convergence very quickly.

Method of representing data transfer and learning according to a data tree. That is, nodes will have limited child nodes. When receiving information, the node only needs to receive data from the child nodes, which allows fewer messages to be sent. forwards until it meets the original node again.

The results below represent each result of the three learning algorithms in order: DHT algorithm, all-to-all algorithm and the new DHT algorithm.

4.2.2 Number of communicated messages

As expected, the number of messages sent by our proposed method is much less than the two previous methods. The protocol differences will be larger if the number of nodes in the Gossip Learning network increases. Details are illustrated in the table 4.5.

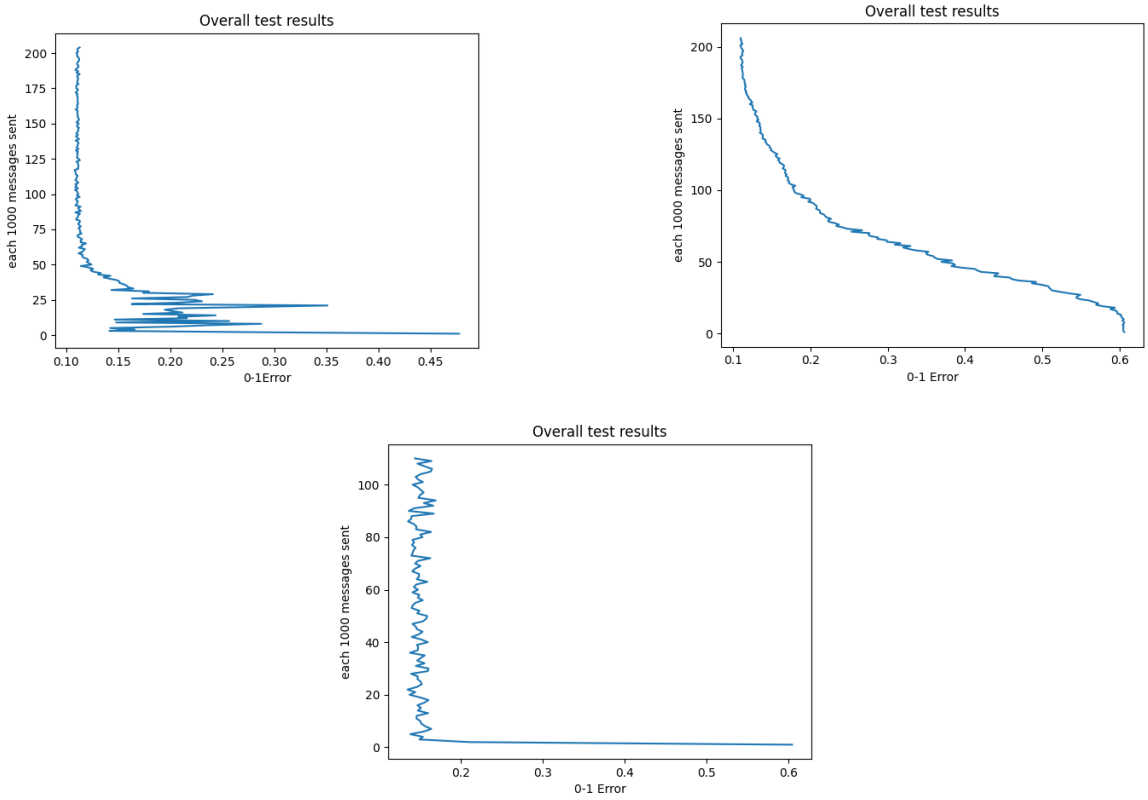


Figure 4.2: 0-1 Error results of All-to-all and DHT-based algorithm

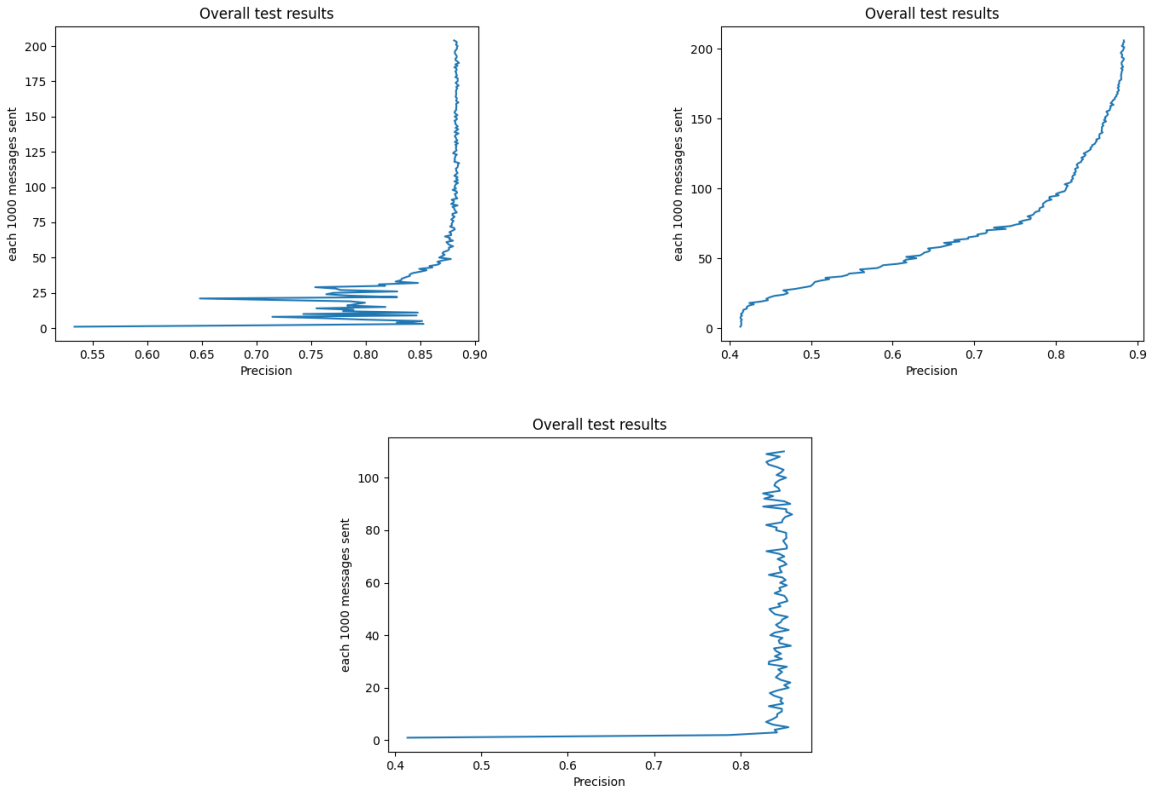


Figure 4.3: Precision results of All-to-all and DHT-based algorithm

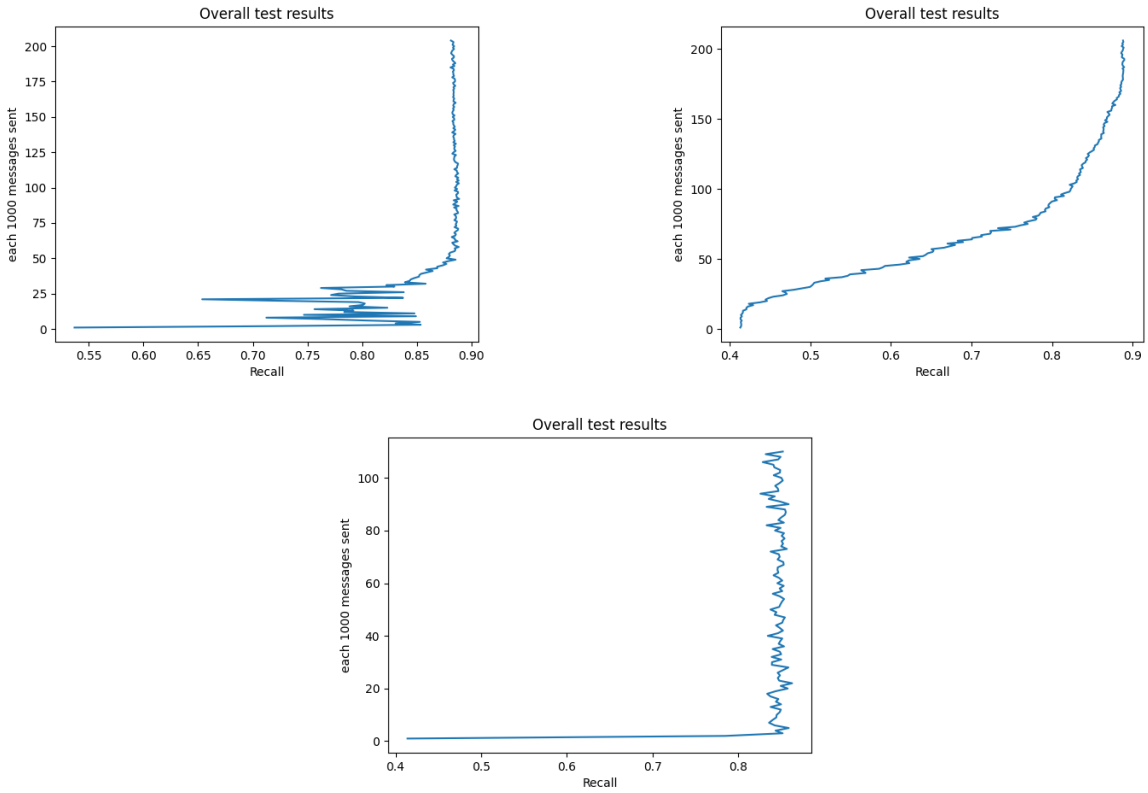


Figure 4.4: Recalll results of All-to-all and DHT-based algorithm

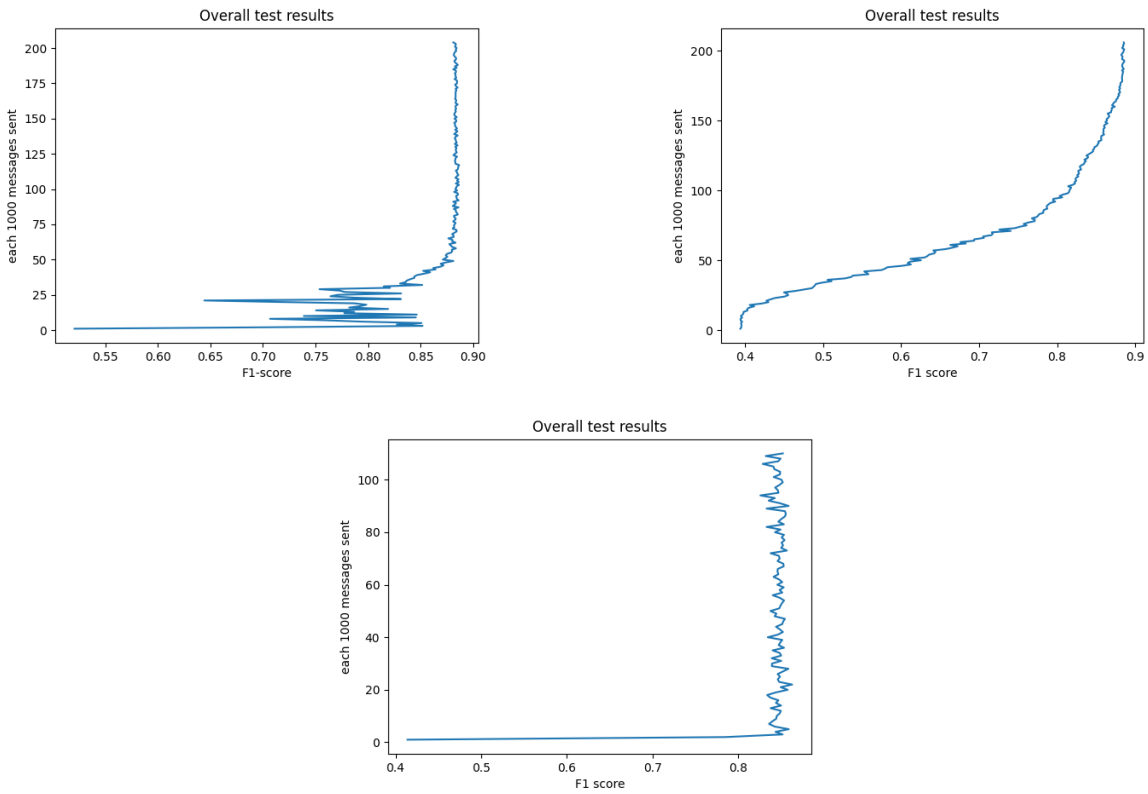


Figure 4.5: F1-Score results of All-to-all and DHT-based algorithm in 4 different seeds

Table 4.4: comparison table of All-to-all, DHT-based protocols and new DHT-based protocols

Protocols	**F1-Score**	**0-1 Error**	**Precision**	**Recall**	**Time running**
All-to-all	0.89	0.11	0.89	0.89	1m42s
DHT-based	0.89	0.11	0.89	0.89	1m42s
New DHT-based	0.88	0.12	0.88	0.88	40s

Table 4.5: Number of sent the messages of two protocols

Seed order	**All-to-all protocol**	**DHT-based protocol**	**New DHT-based protocol**
1	208,800	206,634	22176
2	206,800	204,732	19034
3	203,000	201,718	17266

Conclusion

In summary, this scientific study introduced the use of a new DHT-based protocol in the Gossip Learning network. Our findings demonstrate that the DHT-based protocol effectively reduces the number of neighbors in the Gossip Learning network, reducing the number of messages and running time, which in turn increases system performance. This shows the extremely great benefits of this method in practice compared to the old method.

References

[1] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, "Stochastic gradient push for distributed deep learning," 2019.

[2] M. Cai, K. Hwang, J. Pan, and C. Papadopoulos, "Wormshield: Fast worm signature generation with distributed fingerprint aggregation," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 2, pp. 88–104, 2007.

[3] K. H. Dang, "Collaborative intrusion detection in terminal devices," 2019.

[4] N. B. N. Dung, "A structured peer-to-peer approach for gossip learning," 2023.

[5] R. E. Hopkins, Mark and S. Jaap, "Spambase," UCI Machine Learning Repository, 1999, DOI: https://doi.org/10.24432/C53G6X.

[6] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. U. Stich, "A unified theory of decentralized SGD with changing topology and local updates," *CoRR*, vol. abs/2003.10422, 2020. [Online]. Available: https://arxiv.org/abs/2003.10422

[7] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," 2017.

[8] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2015.

[9] D. C. Nguyen, M. Ding, Q.-V. Pham, P. N. Pathirana, L. B. Le, A. Seneviratne, J. Li, D. Niyato, and H. V. Poor, "Federated learning meets blockchain in edge computing: Opportunities and challenges," 2021.

[10] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multi-task optimization under privacy constraints," 2019.