A Scalable Secure Fault Tolerant Aggregation for P2P Federated Learning

Yujiro Yahata, Keisuke Sugiura and Hiroki Matsutani
Dept. of ICS, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
EMail: {yahata,sugiura,matutani}@arc.ics.keio.ac.jp

***Abstract*—In federated learning (FL), client models rather than the training data are uploaded to the aggregation server, so that clients can participate in a learning process while maintaining their privacy. However, there is still a possibility that the client's data can be inferred from the aggregated model. Also, when the aggregation server fails, the learning process is interrupted. P2P FL systems that eliminate a single point of failure or enable secure aggregation have also been proposed, but they have scalability issues due to the high communication cost. In this paper, we therefore propose a scalable, secure and fault tolerant aggregation system for P2P FL. By introducing a two-layer network which consists of the Secure Average Computation (SAC) layer and Federated Averaging (FedAvg) layer, our system achieves both the privacy protection of participating peers and the reduction of total communication cost. We also propose a fault-tolerant SAC to handle peer dropouts during aggregation. As a backend to the above system, we propose a two-layer Raft, which successfully improves the fault tolerance against random peer crashes. Experimental results show that our two-layer aggregation system outperforms the original SAC in terms of the communication cost and fault tolerance with the comparable accuracy, slow subgroups in SAC layer do not affect the overall accuracy and our two-layer Raft maintains availability by quickly detecting a crashed leader and replacing it with new one. The analysis shows that our system consisting of 30 peers reduces communication costs by 10.36x while providing fault torelance.**

***Index Terms*—privacy-preserving protocols, secure aggregation, machine learning, federated learning, Raft, communication complexity, node scalability, consensus mechanism**

I. Introduction

Federated learning (FL) [1] is a learning method in which each client trains a local model with its own training data and local models are aggregated to update a single global model. In general, a central server updates the global model by aggregating the local models from clients. However, the server becomes a single point of failure, which makes it difficult to continue the federated learning process when the server or the load balancer to which the server belongs fails. Since each client provides a local model rather than data to the central server, FL is a useful technique for privacy protection, but there remains the possibility that the central server could infer the client's training data from the client's model with a membership inference attack [2].

Peer-to-Peer federated learning (P2P FL) has been proposed as one of the solutions to the above problems. In P2P FL, any peer in the network can play the role of an aggregation server. Even if the peer crashes, another peer can take over its role, thus successfully eliminating the single point of failure [3]. In addition, Secure Average Computation (SAC) [4] is a method in which peers compute the aggregated model collaboratively without revealing actual values of their local models. However, SAC has a drawback that the communication cost per aggregation increases quadratically with the number of participating peers, making the system difficult to scale.

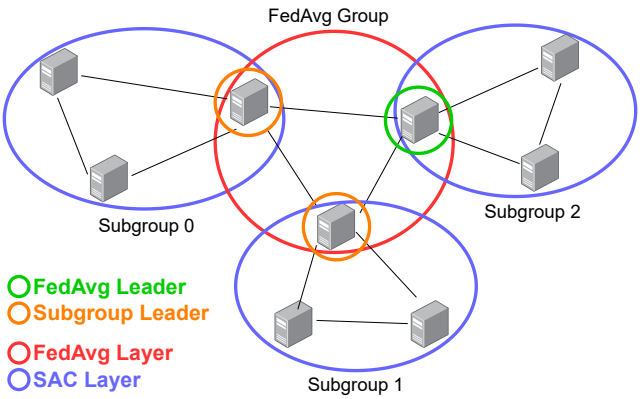

Fig. 1. Network topology of the proposed two-layer aggregation system. In the system, there are $m = 3$ peers in the first layer, and $n = 3$ in each SAC layer subgroup.

In this paper, we propose a scalable, secure and fault tolerant aggregation system for P2P FL. We offer three contributions. First, we build a two-layer aggregation system. As shown in Fig. 1, we divide peers into multiple subgroups, and the system is composed of two layers, namely, SAC layer and FedAvg layer. By combining the SAC layer, which provides privacy protection of each peer, with the communication-efficient FedAvg layer, our system achieves both local model protection for peers and communication cost reduction. Second, we propose a fault-tolerant SAC that allows to continue aggregation even if peers fail. Third, as a backend of our two-layer aggregation system, we propose a two-layer Raft. It provides fault tolerance in the event of random peer crashes. Experimental results demonstrate that our proposed system achieves the accuracy comparable to the original SAC while reducing the communication cost and providing fault tolerance of peers. We also confirm that our two-layer Raft achieves fast crash recovery from various failure cases.

The rest of this paper is organized as below. Sec. II describes the related work for P2P FL, secure aggregation and multi-layer consensus algorithm. Sec. III presents a brief explanation for FL, SAC and Raft consensus algorithm. In Sec. IV, a two-layer aggregation system is proposed and its aggregation flow is described. Sec. V illustrates the system design of two-layer Raft as the backend of the two-layer aggregation system. Evaluation results of the two-layer aggregation system and

the two-layer Raft are shown in Sec. VI. Sec. VII analyzes the communication cost of our aggregation system and fault tolerance of two-layer Raft. Sec. VIII concludes the paper.

II. Related Works

A. P2P Federated Learning

Several methods for P2P federated learning (FL) have been proposed. BrainTorrent [3] is the system to train a shared model in P2P approach, particularly targeting medical applications. A center node in a cluster receives the latest models of other centers and updates its model dynamically. However, semi-honest participants can infer the dataset from weight tensors or inject the false data. Monik et al. [5] showed servers can recover their aggregation network by exchanging messages using asymmetric keys and selecting an aggregation server among them with Raft [6], when the old aggregation server is down. Wink et al. [4] applied an additive secret sharing [7] to P2P FL to ensure that peers obtain the aggregated result while preventing the semi-honest peers from inferring the local dataset from the others' models. However, the communication cost during aggregation increases quadratically with the number of participating peers, which limits the scalability of the system. It also has the disadvantage that if one of the peers is disconnected during aggregation, the aggregation must be restarted from the beginning with remaining peers.

B. Secure Aggregation

Next, we review methods of secure aggregation in normal FL. Recent works have focused on secure aggregation through additive masking [8], [9] to keep the practical performance. In [8], users agree on pairwise secret keys derived by a Diffie-Hellman key exchange protocol, and each user sends masked weight parameters to the server, including pairwise and individual masks. The server can obtain an aggregated weight by summing the masked data received from users because the additive mask sums cancel each other out. This system is tolerant of a certain number of user dropouts, but has the $O(N^2)$ communication cost, where N is the number of users, and suffers from the overhead during the recovery process. Turbo-Aggregate [9] achieves the communication cost reduction of $O(N \log N)$, while maintaining resilience to a 50% user dropout rate. In this method, users are divided into multi groups, which are rotated for each aggregation. The server aggregates the models sent by users in one group and a next group receives this shared model.

C. Multi-layer Consensus Algorithm

As a multi-layer consensus algorithm, a double-layer PBFT model [10] is proposed to reduce the amount of inter-node communications during consensus for blockchain applications. The Practical Byzantine Fault Tolerance (PBFT) consensus algorithm [11] is designed to tolerate Byzantine faults. If we consider the number of nodes capable of tolerating simultaneous faults, denoted as f, then a total of $3f + 1$ nodes are required for the entire network. PBFT offers higher performance than proof based algorithms like Proof-of-Work (PoW) [12]. However, one drawback of PBFT is the low scalability with $O(N^2)$ communication cost for N nodes. To address this issue, the authors of [10] proved that its communication complexity is minimized to about $1.9N^{\frac{4}{3}}$ when nodes are evenly distributed within the second layer subgroups. They also proposed a general X-layer PBFT model and proved that the minimum communication complexity is linear $C = \frac{16N-16}{3}$ when the network depth is maximized.

III. Preliminaries

A. Federated Learning

The general idea of federated learning (FL) is to learn a shared model while keeping the training data on the clients' local devices and avoiding data leakage. Each round involves the following steps. First, clients download a current shared model from a central server and train it using their own data (**local update**). The server collects locally-updated models from randomly selected clients and updates the shared model (**model aggregation**). The new shared model is distributed to the clients for the next round. The above process continues until a target accuracy is achieved.

Federated Averaging (FedAvg) [1] is widely used as an aggregation method. Let η be a fixed learning rate, g_k be the gradient update from k-th client and w_t be the shared model at t-th round. An updated shared model, denoted as w_{t+1}, is given by $w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{N} \frac{n_k}{n} g_k$, where n_k is the number of training samples used in the client k, N is the number of clients selected for the aggregation and n is the sum of n_k. This means the model w_t is updated by the sum of local gradients weighted by the number of samples. With the updated model for each client, denoted as $\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k$, the above update law is rewritten as $w_{t+1} \leftarrow \sum_{k=1}^{N} \frac{n_k}{n} w_{t+1}^k$.

B. Secure Average Computation (SAC)

Secure Average Computation (SAC) [4] is an approach to collaboratively train a shared model in a decentralized manner without sharing the actual model (w_{t+1}^k in Sec. III-A). A simple n-out-of-n secret sharing scheme, especially additive secret sharing [13], is a basic form of SAC to calculate the sum of models of all peers. The aggregation process in each round is presented in Alg. 2. In the first phase, the model for the i-th peer wt_i is randomly divided into N shares $\mathrm{par\_wt}_i = (\mathrm{par\_wt}_{i1}, \ldots, \mathrm{par\_wt}_{iN})$, that satisfy $\mathrm{wt}_i = \sum_{j=1}^{N} \mathrm{par\_wt}_{ij}$ (Alg. 1). The peer i sends $\mathrm{par\_wt}_{ij}$ to the corresponding peer j, where $1 \leq j \leq N, j \neq i$ (line 3) and keeps the remaining $\mathrm{par\_wt}_{ii}$ share. The peer i receives the shares $\mathrm{par\_wt}_{ji}$ $(1 \leq j \leq N, j \neq i)$ from the other $N-1$ peers (line 5). With these shares, the subtotal $\mathrm{ps\_wt}_i$ is calculated (line 6), which is broadcast to all other peers (line 7). After receiving subtotals from all peers $(\mathrm{ps\_wt}_1, \ldots, \mathrm{ps\_wt}_N)$, the peer i computes their average w_{avg} (line 9), which is identical to the original average since w_{avg} is rewritten as:

$$w_{avg} = \frac{\Sigma_{j=1}^{N} \mathrm{ps\_wt}_j}{N} \quad (1)$$

$$= \frac{\Sigma_{j=1}^{N} \Sigma_{i=1}^{N} \mathrm{par\_wt}_{ij}}{N} \quad (2)$$

$$= \frac{\Sigma_{i=1}^{N} \mathrm{wt}_i}{N}. \quad (3)$$

Algorithm 1 Algorithm to split a secret into N shares

Require: number of peers N, secret to be split w

function Divide(N, w)
  rn = Array of N random numbers $(\mathrm{rn}_1, \mathrm{rn}_2, \ldots, \mathrm{rn}_N)$
  prn = Array of length N $(\mathrm{prn}_1, \mathrm{prn}_2, ..., \mathrm{prn}_N)$
  for $i = 1$ to N **do**
    $\mathrm{prn}_i = \frac{\mathrm{rn}_i}{\Sigma_{k=1}^{N} \mathrm{rn}_k}$
  par_w = Array of length N $(\mathrm{par\_w}_1, \mathrm{par\_w}_2, \ldots, \mathrm{par\_w}_N)$
  for $i = 1$ to N **do**
    $\mathrm{par\_w}_i = \mathrm{prn}_i \cdot w$
  return par_w

Algorithm 2 SAC in peer i $(1 \leq i \leq N)$

Require: number of peers N, model wt_i

function SecAVG(N, wt_i)
  $\mathrm{par\_wt}_i = \mathrm{Divide}(\mathrm{wt}_i, N)$ ▷ dim: number of matrix dimensions
  for $j = 1$ to N **do** ▷ Send a share of each model to other peers
    Send $\mathrm{par\_wt}_{ij}$ to peer_j
  Save all received partitions and $\mathrm{par\_wt}_{ii}$
  $\mathrm{ps\_wt}_i = \Sigma_{j=1}^{N} \mathrm{par\_wt}_{ji}$ ▷ Calculate subtotals of each model
  Send $\mathrm{ps\_wt}_i$ to all peers
  Save all received subtotals and $\mathrm{ps\_wt}_i$
  $w_{avg} = \frac{\Sigma_{j=1}^{N} \mathrm{ps\_wt}_j}{N}$
  Confirm if all other nodes successfully finished SAC
  return w_{avg}

Here, we analyze the communication cost of SAC in each round. Let $|w|$ be the size of a weight tensor of a model w. First, the communication cost of the whole network to exchange shares $\mathrm{par\_wt}_{ij}$ is given by $N(N-1)|w|$ (lines 3–4). Next, the cost to exchange the subtotals $\mathrm{ps\_wt}_i$ of size $|w|$ is $N(N-1)|w|$ (line 7). The total cost for an aggregation is $2N(N-1)|w|$. This means SAC has limited scalability since the cost increases quadratically with the number of peers.

C. Raft Consensus Algorithm

Consensus algorithms are typically built upon a replicated state machine [14] that allows a group of machines to remain coherent and survive the failures of some of its members. They ensure the safety under practical conditions and the availability as long as majority of the machines are running. They also keep the consistency of the state machine regardless of timing. The overall system performance is typically not affected by a few slow servers.

Raft [6] is a consensus algorithm that has similar performance to others like Paxos [15] under non-Byzantine conditions and successfully enhances understandability of the algorithm. Raft enables a crashed server to rejoin the cluster at any time by following the state transition sequence (Fig. 2). Raft decomposes the consensus problem into three relatively independent subproblems: leader election, log replication and safety.

1) Three States of Servers and Leader Election: As shown in Fig. 2, each server is in one of three states: leader, follower and candidate. The leader handles all requests from clients, and followers respond to requests from the leader or candidates. Candidates are the servers which could be elected as a leader.

Raft has a logical clock called term, which is a sequence of integers. Each term starts with a leader election. If a candidate receives votes from more than a half of the servers in a cluster, it becomes the leader for the rest of the term. The leader regularly sends heartbeats to all followers to inform that the leader is present. If a follower does not receive heartbeats for a certain period of time, called the election timeout, it assumes there is no valid leader and initiates an election. To start the election, the follower increments its term, changes its state to candidate, and invokes RequestVote RPCs to gather votes from other servers.

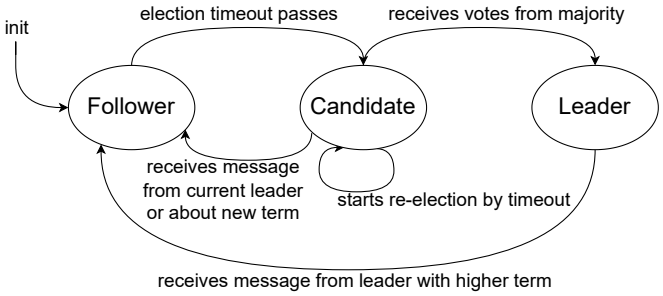

Fig. 2. Server states in Raft consensus algorithm. A server in Raft transitions among three states: leader, follower and candidate.

2) Log Replication: The elected leader accepts requests from clients and executes commands contained in requests for its replicated state machine. It appends the command to its log as a new entry and then issues AppendEntries RPCs to replicate the same entry in all servers. Once the entry has been safely replicated (committed) on a majority of the servers, the leader applies the command to its state machine and returns the execution result to the client. Committed entries are durable, meaning the command in the entry will eventually be executed on all available state machines. The leader keeps track of an index of the latest committed entry and includes it in AppendEntries RPCs to synchronize the follower's state with the leader. Raft keeps the log coherency across different servers, while satisfying the following properties: if two different logs have an entry with the same index and term, they are identical up to that index.

3) Safety: From the viewpoint of safety, we describe the restrictions in the election and the case of committing entries of previous terms. First, Raft imposes a restriction on the leader, i.e., it should keep all the entries committed in the previous terms at any point. For this reason, only a candidate with the up-to-date log wins the election. During the election, RequestVote RPCs sent by the candidate include the candidate's log status so that the voter can determine whether the candidate keeps its log up-to-date. There is also no possibility that the leader commits an entry for a past term (only entries for the current term are committed). This means all previous entries are present in the log when an entry for the current term is committed.

IV. Two-Layer Aggregation System

We propose a two-layer model parameter aggregation system to improve the communication efficiency, security and fault-tolerance of the P2P federated learning. This section presents the flow of aggregation, how to select a leader to take the initiative in aggregation, and the details of aggregation in each layer.

A. Aggregation Flow

Fig. 1 shows the overview of our system. The peers are divided into multiple subgroups and the system is composed of two layers, namely, SAC layer and FedAvg layer. SAC layer corresponds to each subgroup (blue circles) and FedAvg layer is for a higher group (red circle). Each SAC subgroup has a leader (orange circles) and FedAvg layer also has a leader called the FedAvg leader (green circle). As shown in Sec. III-B, SAC lacks the scalability due to $O(N^2)$ communication cost, but that of our system is reduced from $O(N^2)$ to $O(nN)$ as described later.

Each aggregation is performed in the following steps (Alg. 3). First, a FedAvg leader requests subgroup leaders in SAC layer to upload SAC-aggregated models. Upon receiving the request, each subgroup leader performs SAC (line 3). When SAC is done, the subgroup leader sends aggregated models to the FedAvg leader (line 9). The FedAvg leader starts aggregation via FedAvg with enough amount of models collected from subgroups (line 10). The aggregated result is broadcast to all subgroup leaders. All peers in subgroups receive it from their corresponding subgroup leader and resume learning with their own data. Our proposal is agnostic to the aggregation algorithm, which can be chosen appropriately for each use case.

Algorithm 3 Two-layer SAC

Require: A set of subgroup leaders S
1: **function** ExecuteTwoLayerSAC(S)
2: $W \leftarrow \varnothing$ ▷ A set of aggregated models W for subgroups
3: **for each** leader$_i \in S$ **do**
4: n = length(leader$_i$) ▷ leader$_i$ checks the number of peers in its subgroup
5: k = takeThreshold(leader$_i$) ▷ leader$_i$ picks up a threshold for reconstructing secret weights
6: $\mathrm{wt}_i \leftarrow$ getClientModel ▷ a model of leader$_i$
7: $w^i_{avg} \leftarrow$ FaultTolerantSAC(n, k, wt_i) ▷ leader$_i$ initiates k-out-of-n SAC
8: $N \leftarrow N \cup \{n\}$
9: $W \leftarrow W \cup \{w^i_{avg}\}$ ▷ An updated set of aggregated models by leader$_i$
10: $w_{avg} = \frac{\sum_{w \in W, n \in N} w \cdot n}{\sum_{n \in N} n}$ ▷ FedAvg
11: **return** w_{avg}

B. Aggregation Leader

In our system, each subgroup has one leader elected by Raft and multiple followers. Generally, the leader takes the initiative in aggregation and followers respond to the requests from their leader. Followers participate in the Raft process and acquire the information about their leader on their own. The follower discards the aggregation request if the sender is different from the current leader tracked by the follower itself. Subgroup leaders in SAC layer are connected to FedAvg layer via an additional Raft process. FedAvg layer also has one leader, which aggregates the models from the other followers (i.e., subgroup leaders) to update a shared model and proceed to the next round (Alg. 3).

C. SAC Layer with Fault-Tolerance

During SAC, the models of subgroup peers are aggregated via a simple n-out-of-n additive secret sharing on a complete graph network, without each peer having to share its model to others. However, one drawback is that SAC is not robust to random failures. Even if one peer is disconnected, the aggregation must be aborted. In Alg. 2, if the k-th peer fails to share its partitions $\mathrm{par\_wt}_{kl}$ $(k \neq l)$ (line 4), the other peers cannot compute the subtotal $\mathrm{ps\_wt}_m$ $(m \neq k)$ due to the missing partition, and thus the aggregation process cannot proceed. To solve this problem, we apply Replicated Additive Secret Sharing [7] like k-out-of-n $(1 \leq k \leq n)$ to SAC to allow failures of up to $n-k$ nodes. k-out-of-n SAC means the aggregation is still operational as long as k out of n peers are alive.

Alg. 4 shows our proposed aggregation method in each round. Let N be the total number of peers participating in SAC, K $(1 \leq K \leq N)$ be the threshold for reconstructing the secret average and wt_i be the model for the leader (i-th peer). First, after randomly partitioning wt_i into N shares (line 2), the i-th peer sends $N-K+1$ consecutive partitions to each peer (lines 3-9). The i-th peer receives the partitions (line 10) and then starts calculating $N-K+1$ subtotals $\mathrm{ps\_wt}_{j'}$ $(j' = j \bmod N,\ i \leq j \leq i+N-K)$ (lines 11–13). If peer i is the leader, it receives subtotals from other peers which it does not contain (lines 14–16). When the leader detects the peer j $(j \neq i)$ is down, it asks $\mathrm{peer}_{k \bmod N}$ $(j-(N-K) \leq k \leq j-1)$ for $\mathrm{ps\_wt}_j$ (lines 17–18). After receiving all subtotals (line 19), the leader computes the average of them w_{avg} (line 20). A diagram in Fig. 3 presents the specific 2-out-of-3 SAC case, where though one peer drops out during the aggregation, the remaining peers complete the SAC.

Algorithm 4 Fault-tolerant SAC in the leader (peer i) $(1 \leq i \leq N)$

Require: number of peers N, threshold for reconstructing secret weights K $(1 \leq K \leq N)$, weight tensors wt_i
1: **function** FaultTolerantSAC(N, K, wt_i)
2: $\mathrm{par\_wt}_i = \mathrm{Divide}(\mathrm{wt}_i, N)$
3: **for** $j = 0$ to $N-1$ **do** ▷ Send shares of weight values to other peers
4: pars $\leftarrow \varnothing$ ▷ A set of shares pars to be sent to peer$_j$
5: **for** $k = j$ to $j+N-K$ **do**
6: $j' \leftarrow j \bmod N$
7: pars $\leftarrow$ pars $\cup \{\mathrm{par\_wt}_{ij'}\}$
8: **if** $j \neq i$ **then**
9: Send pars to peer$_j$
10: Save all received partitions and $\mathrm{par\_wt}_i$
11: **for** $j = i$ to $i+N-K$ **do**
12: $j' \leftarrow j \bmod N$
13: $\mathrm{ps\_wt}_{j'} = \Sigma_{k=0}^{N-1} \mathrm{par\_wt}_{kj'}$ ▷ Calculate subtotals of each weight
14: **for** $j = i-K+1$ to $i-1$ **do**
15: $j' \leftarrow j \bmod N$
16: Receive $\mathrm{ps\_wt}_{j'}$ from peer$_{j'}$
17: **if** peer$_{j\ (0 \leq j \leq N-1)}$ is crashed **then**
18: Ask peer$_{k \bmod N\ (j-(N-K) \leq k \leq j-1)}$ for $\mathrm{ps\_wt}_j$
19: Save received subtotals and $\mathrm{ps\_wt}_{j \bmod N\ (i \leq j \leq i+(N-K))}$
20: $w_{avg} = \frac{\Sigma_{j=1}^{N} \mathrm{ps\_wt}_j}{N}$
21: Confirm if all other nodes successfully finished SAC
22: **return** w_{avg}

D. FedAvg Layer

A FedAvg leader aggregates the models from SAC layer via FedAvg. Considering that SAC is employed in the lower layer

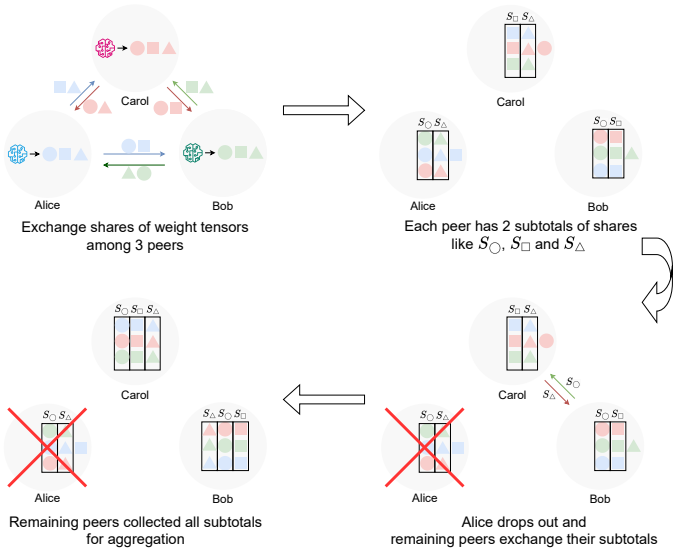

Fig. 3. Situation where one peer drops out during 2-out-of-3 SAC. In the first state (upper left), all three peers are exchanging shares of their own weight tensors. In the second state (upper right), Each peers contain two subtotals of shares like $S_{\bigcirc}$, $S_{\square}$ and $S_{\triangle}$. In the third state (bottom right), remaining two peers still exchange their subtotals even though Alice drops out. In the last state (bottom left), remaining peers successfully collect all subtotals required for aggregation.

and the private training data of the peers cannot be recovered directly from the SAC-aggregated models, the standard FedAvg can be used instead of the secure aggregation method in the higher layer. However, security analysis for the aggregated model is out of the scope in this paper. In case where stronger privacy guarantees are needed, SAC could be employed in the higher layer. Other techniques such as Differential Privacy [16] could be used to add noise to the weight of each peer.

V. Two-Layer Raft

We propose two-layer Raft, which serves as the backend of our two-layer aggregation system. The main reason for combining Raft with P2P federated learning is to ensure fault tolerance in the event of an aggregation leader crash. The Raft leader election can quickly recover the whole system after the leader crash and helps to continue the aggregation process. Log replication is only used to share the configurations of FedAvg layer with SAC layer subgroups. This section describes how our system recovers from the four failure cases, i.e., the failure of leader/follower in FedAvg/SAC layer.

A. Fault Tolerance in SAC Layer

1) Leader Crash: If the subgroup leader crashes, Raft leader election is initiated by the remaining peers in that subgroup. Additionally, our system has a post-leader election callback, where the newly elected leader automatically joins the group in FedAvg layer as a follower by referring to the FedAvg-layer configuration. The whole procedure of leader crash recovery is presented in Fig. 4. First, followers in the subgroup detect the leader crash from not receiving the heartbeat. Some of them become candidates for the next term when their own election timeout elapses. Candidates send RequestVote RPCs to other peers in order to receive votes

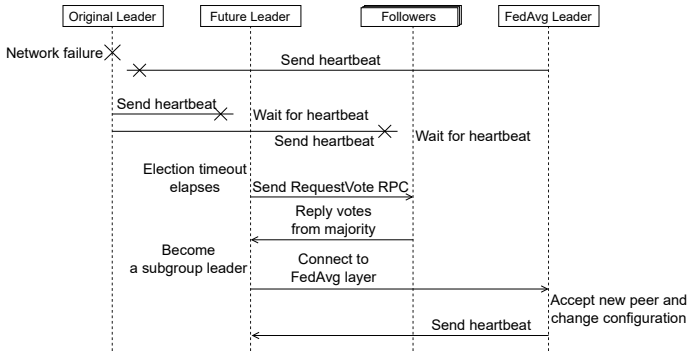

Fig. 4. Overview of a SAC-layer leader crash recovery

from the majority before anyone else. Once the candidate becomes a leader in that subgroup, it will automatically join FedAvg layer as a follower. In our system, the subgroup leader periodically commits the configuration of FedAvg layer, e.g., IP addresses and IDs of peers in FedAvg layer, to its subgroup state machine.

When Raft is operational in FedAvg layer, the majority of peers stored in the FedAvg-layer configuration are expected to be still alive, so a subgroup leader can connect to the FedAvg leader directly or through other FedAvg-layer followers. Thanks to the strong consistency in Raft, once configurations are committed, it is guaranteed to be shared by the majority of state machines in the subgroups. As described in Sec. III-C, the peer is elected as a leader only if it has the latest configuration in the log.

2) Follower Crash: The Raft network is tolerant to the random crash of the followers and keeps running as long as a majority of the peers are active.

B. Fault Tolerance in FedAvg Layer

1) Leader Crash: If the FedAvg leader crashes, two leader elections take place because it was the subgroup leader in SAC layer as well. First, a leader election is performed with the remaining peers in FedAvg layer. At the same time, the election is performed at the subgroup in SAC layer as well. Note that the new subgroup leader cannot join FedAvg layer until the election in FedAvg layer completes, because a new peer cannot be added to the FedAvg-layer group without the FedAvg leader present. A subgroup leader periodically sends a request to FedAvg layer to confirm the presence of a FedAvg leader. Once a FedAvg leader is detected, a subgroup leader is connected to FedAvg layer following the procedure similar to that described in Sec. V-A1.

2) Follower Crash: Since the follower in FedAvg layer is also a leader in a SAC-layer subgroup, its crash leads to an unstable state where no leader is present in the subgroup. As shown in Sec. V-A1, a leader election in the subgroup is initiated and the new leader connects to FedAvg layer.

VI. Implementation and Evaluation

We implemented a two-layer parameter aggregation for P2P federated learning based on a two-layer Raft. Our system is mainly divided into two parts. The first one is a federated learning part that handles the model training and two-layer

TABLE I
EVALUATION ENVIRONMENT

OS	Ubuntu 20.04 LTS
CPU	Intel Core i9-10980XE @ 4.6GHz
GPU	NVIDIA GeForce RTX 3090Ti 24GB
DRAM	DDR4 DRAM 128GB

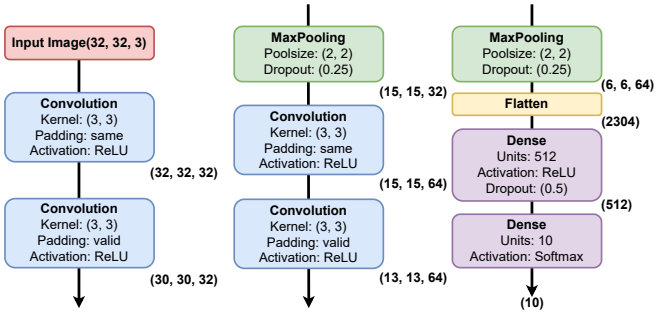

Fig. 5. CNN model architecture (same as in the baseline [4]) for CIFAR-10. The model is relatively small with 1.25M parameters. It consists of two building blocks, each having two consecutive convolutional layers followed by max pooling and dropout. ReLU is used as an activation function. Two dense layers are placed before the output. The first dense layer has ReLU and dropout, while the second one has softmax activation.

parameter aggregation on a P2P network. The second one is a Raft part for a leader election, which is a key to the fault tolerance. It allows to continue the federated learning after random peer crashes by dynamically assigning the leader role to remaining alive peers. The federated learning part uses Python 3.8.10 and PyTorch 2.0.1. The two-layer Raft is implemented with Go 1.21.4 and hashicorp/raft 1.5.0 [17]. Our system is an extension of hashicorp Raft, with gRPC [18] used for communication between FedAvg and SAC layers. The machine in Table I is used throughout the evaluation.

A. Two-Layer Aggregation

1) Evaluation Setup: CNN model used in this evaluation is the same as the one in the baseline P2P-FL [4], and its detailed structure is presented in Fig. 5.

We used two datasets (MNIST [19] and CIFAR-10 [20]) for image classification in our two-layer aggregation. MNIST is a well-known set of grayscale handwritten digit images (28x28 pixels). The dataset is divided into a training set of 60,000 labeled images and a test set of 10,000 labeled images. The CIFAR-10 dataset includes 60,000 labeled RGB images of 32x32 pixels with 50,000 for training and 10,000 for testing. As for the training data distribution, three different cases are considered as in the paper [4].

- **IID data**: Training dataset for each peer is identically and independently distributed.
- **Non-IID data** (5%): The 95% of training data for each peer are samples from two main classes randomly selected out of the ten classes, and the 5% are taken from the rest classes.
- **Non-IID data** (0%): Training dataset for each peer contains samples from two main classes randomly selected out of the ten classes.

We utilized the *Adam* optimizer with a learning rate of 0.0001 and categorical cross-entropy as a loss function. The number

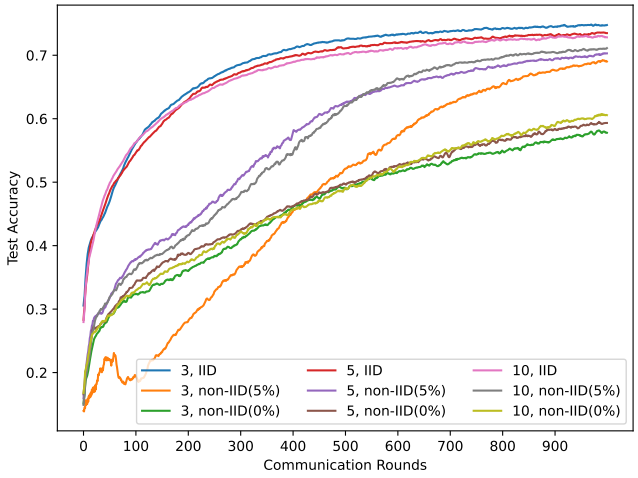

Fig. 6. Moving average of test accuracy between two-layer SAC and the baseline for a total of $N = 10$ peers and 1000 communication rounds. Each subgroup has $n = 3, 5, 10$ peers. The peers are equally divided into subgroups. For example, in case of $n = 3$, the $N = 10$ peers are divided into three subgroups with 3, 3, and 4 peers each. In case of $n = N$, the system is simplified to the original SAC.

of training rounds is set to 1000, the number of epochs in each round to 1, and the batch size to 50. In each round, peers train their models with their own datasets, and then the models are aggregated via SAC in subgroups. The SAC-aggregated models are then aggregated by the FedAvg leader. Finally, the aggregated model in FedAvg layer is then sent to all peers in SAC layer and the next round starts.

2) Accuracy and Loss for CIFAR-10: First, we compared original SAC (Alg. 2) to our proposed two-layer SAC (Alg. 3) in terms of the test accuracy and training loss. Figs. 6–7 show the results. In these evaluations, the total number of peers is $N = 10$, and every subgroup is formed by $n = 3, 5, 10$ peers. The $n = 10$ case (i.e., only one subgroup) corresponds to the original SAC. As shown in the figures, our two-layer SAC (blue, red) has the same test accuracy and training loss as the baseline (pink) while reducing the communication complexity. Both methods achieve the best accuracy under the IID data setting, and the accuracy drops as the data distribution deviates from the IID. After 1000 rounds, the $n = 3$ case achieves the highest accuracy of 74.69% under IID data setting. On the other hand, the accuracy of $n = 3$ is 57.95% under non-IID (0%) setting. Differences in the number of peers in subgroups do not cause significant accuracy changes (less than 2% in most cases). The similar results are obtained with MNIST dataset as well.

3) Resilience to Slow Subgroups: Next, we show slow subgroups in SAC layer do not affect the overall accuracy with CIFAR-10 dataset. In this evaluation, only m out of $\lfloor N/n \rfloor$ subgroups participate in the FedAvg aggregation to simulate slow subgroups that cannot submit their SAC-aggregated models to the FedAvg leader before timeout. We denote $p = m/\lfloor N/n \rfloor$ as a fraction of the subgroups used. Figs. 8–9 show the accuracy and loss under $N = 20$, $n = 5$, and $p = 0.5, 1$. Here, $p = 1$ means the FedAvg leader aggregates models of all subgroups and $p = 0.5$ means the FedAvg leader aggregates half of the models from subgroups. Since the number of subgroups is four in this condition, two subgroups

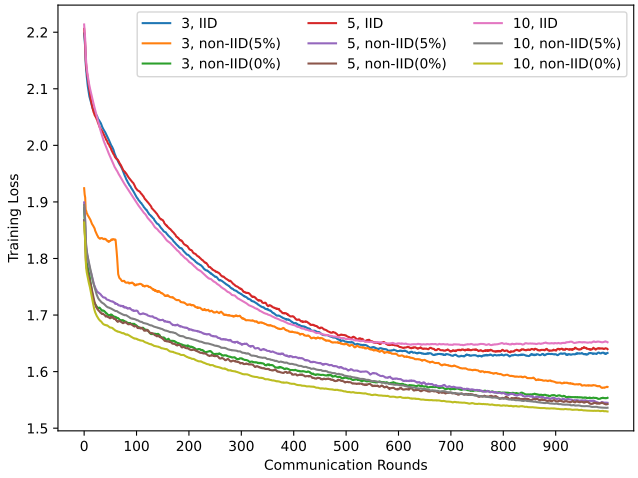

Fig. 7. Moving average of training loss between two-layer SAC and the baseline for $N = 10$. The experimental setting is the same as that in Fig. 6.

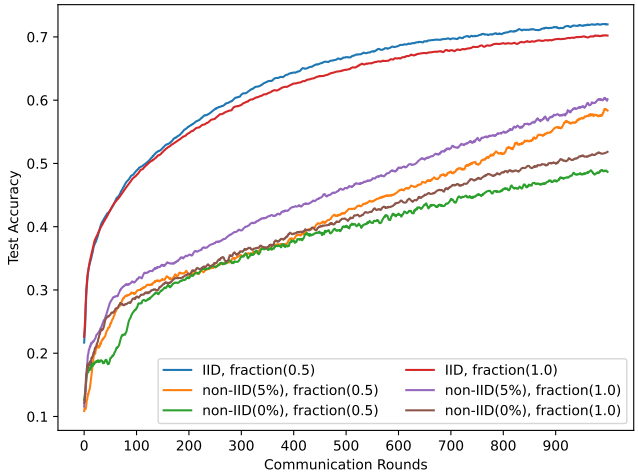

Fig. 8. Moving average of test accuracy for a different fraction in the two-layer SAC ($N = 20,\ n = 5$). A fraction is the proportion of SAC layer subgroups used for the FedAvg layer aggregation.

are involved in the FedAvg layer aggregation.

Fig. 8 confirms that the similar accuracy is obtained even though the FedAvg leader cannot collect models from all subgroups. In fact, the average of accuracy differences between $p = 0.5$ and 1 is 2.18% in three different cases of the data distribution. Thus, the FedAvg layer leader does not necessarily receive responses from all subgroups in each aggregation, and timeout will help the global model converge faster. The IID data setting gives the best accuracy as in Fig. 6 and the result shows an accuracy drop in the Non-IID data settings. In case of IID setting, the accuracy of $p = 0.5$ is 72.09% after 1000 rounds. On the other hand, the lowest accuracy of 48.69% is obtained with $p = 0.5$ and Non-IID (5%) settings.

B. Two-Layer Raft

1) Evaluation Setup: In this evaluation, we execute two-layer Raft on a single machine as shown in Table I. We built a simulation environment with a configurable number of virtual peers. Each virtual peer has its own IP and ID, for FedAvg/SAC layers to simulate the FedAvg/SAC-layer leader crash and communicates using TCP on the same machine.

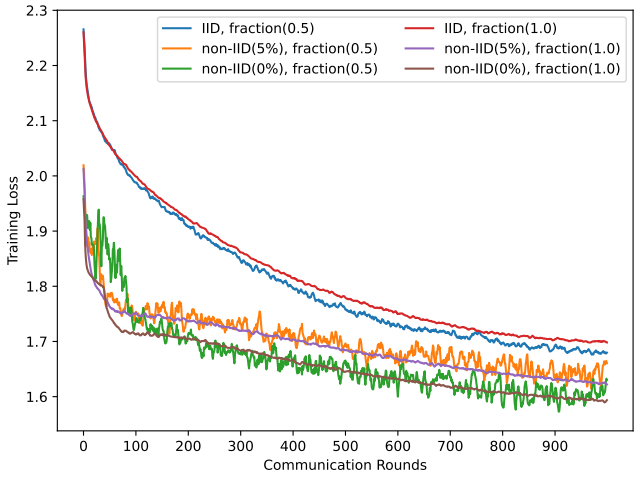

Fig. 9. Moving average of training loss for a different fraction in 2-Layer SAC ($N = 20,\ n = 5$). The experimental setting is the same as that in Fig. 8.

The tc command [21] is used to simulate network latency. The delay is 15ms, which is the same as in [6] and [22]. We also follow [6] to set the timeout parameters as below.

$$broadcast\ time \ll candidate\ timeout \ll MTBF$$
$$candidate\ timeout = follower\ timeout \sim U(T, 2T)$$

Here, each peer has its own *candidate timeout* and *follower timeout*, which mean required timeouts to remain a candidate and a follower without the contact from a leader, respectively. The peer starts an election when the timeout is over. The mean time between failures (*MTBF*) indicates the time that elapses between failures of a single server and *broadcast time* is the average time for a server to simultaneously send RPCs to all servers in the cluster and receive their responses. $U(T, 2T)$ is a value between T and $2T$ taken from uniform distribution ($T = 50, 100, 150, 200$). There are five subgroups each with five peers ($N = 25,\ n = 5$).

2) Recover from A Crashed Subgroup Leader: We show the time to detect a leader crash and elect a new leader in a subgroup (Sec. V-A1). We run the simulation 1000 times for each range of timeout. Here, we have not taken into account the worst case scenario where the existence of the peer whose log is outdated. This is because the value updated in the subgroup log is only the configuration of the FedAvg layer, which is not large.

Fig. 10 shows the results. As expected, it takes longer to elect a leader with a larger *follower timeout*. The average is 214.30ms, 401.04ms, 580.74ms and 749.07ms in case of 50–100ms, 100–200ms, 150–300ms and 200–400ms, respectively, which is about twice the maximum *follower timeout*. This is because the *follower timeout* is the time required for the follower to regard the leader as absent, and the follower cannot become a candidate until that time has passed. However, if the *follower timeout* is too short, it is more likely that followers will misunderstand the leader's absence and elections will take place frequently. In fact, even when a peer become a leader, its authority was not stable and elections were held repeatedly in case of 12–24ms.

Fig. 11 shows the time to detect a crashed subgroup leader

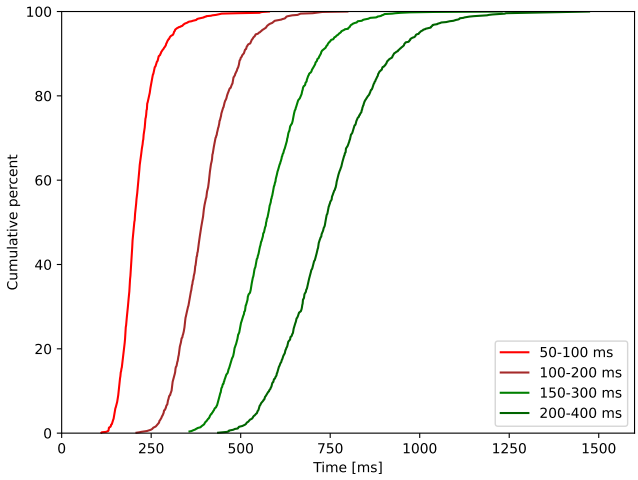

Fig. 10. Time to detect a crashed leader and elect a new leader in a subgroup. *follower timeout* for each peer is randomly sampled from a uniform distribution with four different ranges (in the legend). *candidate timeout* is set the same value as *follower timeout*. Experiments were conducted 1000 times per each setting.

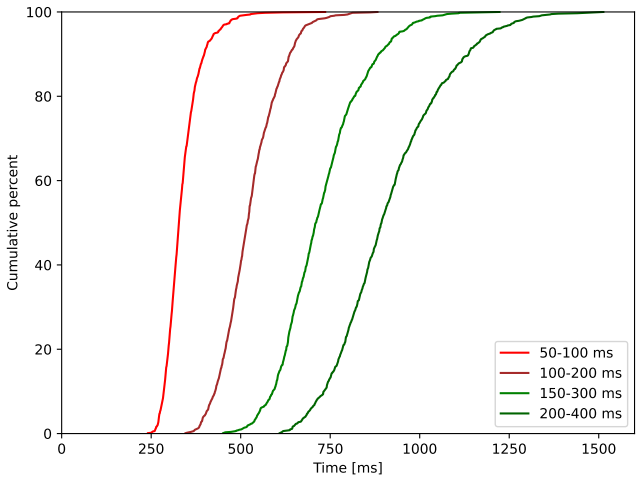

Fig. 11. Time to detect a crashed leader, elect a subgroup leader and for that leader to join the FedAvg group. The experimental setting is the same as that in Fig. 10.

and for a new leader to join the FedAvg group. Compared to Fig. 10, the average time takes longer by 122.98ms, 125.8ms, 144.70ms and 166.09ms in case of 50–100ms, 100–200ms, 150–300ms and 200–400ms, respectively. In our system, the peer is automatically connected to FedAvg layer when elected as a new subgroup leader. Thus, Fig. 11 shows the similar trend as in Fig. 10. The difference is that Fig. 11 considers an additional time required to join the FedAvg layer. The downtime is still expected to be far less than one round of federated learning.

3) Recover From A Crashed FedAvg Leader: We confirm that our system can recover from a crash of the leader of both FedAvg group and subgroup (Secs. V-A1 and V-B1). The newly elected subgroup leader is then connected to the FedAvg group. If a leader has not been elected in the FedAvg group yet, the subgroup leader has to wait until a FedAvg leader is set. In this experiment, the subgroup leader checks the presence of the FedAvg leader every 100ms, and *follower timeout* is sampled from an uniform distribution for both the FedAvg

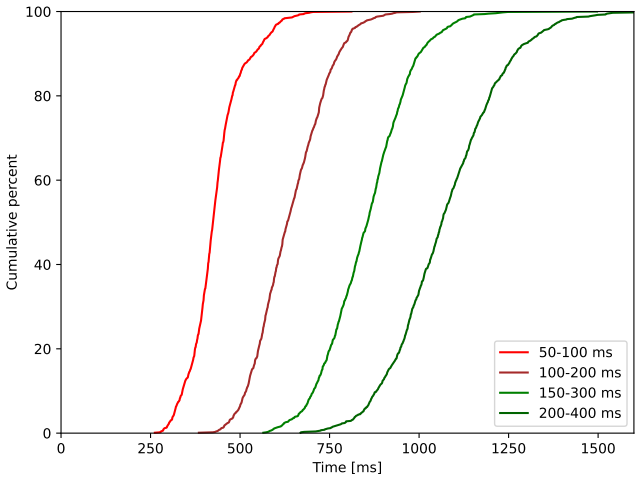

Fig. 12. Time to detect crashed leaders of both the FedAvg group and subgroup, elect leaders of both the FedAvg group and subgroup, and for new subgroup leaders to join the FedAvg group. The experimental setting is the same as that in Fig. 10.

group and subgroup.

Fig. 12 shows the time to recover the system (i.e., detect the FedAvg leader crash, elect new leaders of both the FedAvg group and subgroup and rebuild the FedAvg group with new subgroup leaders). The average time required is longer than in the previous case (Fig. 11) : 95.07ms, 114.65ms, 130.30ms and 158.53ms for the four timeout settings, respectively, because the recovery of the entire network requires longer time than that of one subgroup. The new subgroup leader has to wait the completion of an election in the FedAvg group, which also affects the performance. In this case, the interval can be set less than 100ms to connect to the FedAvg group faster. The simulation results indicate that our system ensures sufficient availability even if the FedAvg leader crashes.

VII. Analysis for Two-Layer Aggregation and Raft

A. Communication Analysis for Two-Layer Aggregation System with n-out-of-n Secret Sharing

The main reason for breaking down the aggregation into two layers is to reduce the communication cost in a whole network while keeping the private local model on each peer secret. One major problem in SAC is its low scalability due to the $O(N^2)$ communication cost, especially $2N(N-1)|w|$, where N is the number of peers as mentioned in Sec. III-B and $|w|$ is the size of the model w. On the other hand, the communication cost of our proposed two-layer aggregation is reduced. The cost for FedAvg is $2(m-1)|w|$, the one for broadcasting the FedAvg-aggregated weight to all SAC peers is $m(n-1)|w|$ and the cost for SAC of all m subgroups is $m(n^2-1)|w|$, since n peers exchange $n-1$ partitions and then $n-1$ followers send a subtotal to the leader. Therefore, the total communication cost per aggregation is

$$(mn^2+mn-2)|w|. \tag{4}$$

Since the total number of peers in the system is given as $N = nm$, a larger number of groups m leads to a smaller group

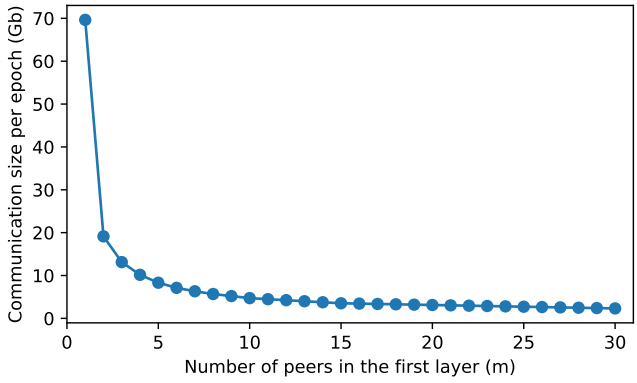

Fig. 13. Total communication cost per aggregation with a varying m. We use the CNN model as described in Fig. 5. Note that FedAvg layer consists of m subgroup leaders. The total number of peers is fixed to $N = 30$. N/m peers are assigned to each subgroup. When remaining $N \bmod m$ peers exist, these are distributed to the subgroups as evenly as possible. For instance, in case of $N = 30$ and $m = 4$, the system has four subgroups, two of which contain eight peers and the others contain seven peers.

size n. As a result, the overall cost is reduced from $O(N^2)$ to $O(nN)$. From the peer's perspective, the cost is reduced from $O(N)$ to $O(n)$.

Fig. 13 shows the communication cost per aggregation under a varying m. Note that the aggregation is simplified to the standard FedAvg without SAC when $m = N$, and to the original one-layer SAC without FedAvg when $m = 1$. When $m = 6$, the communication cost is 7.12Gb, which means the cost is about one-tenth of that of the one-layer SAC. When m is greater than 10, some subgroups consist of fewer than three peers, which means SAC is no longer secure and the backend raft is not fault tolerant. In case of $n = 2$, the weight of the other peer can be easily inferred, meaning that n should be larger than three. In addition, the communication cost does not decrease when $m \geq 10$ ($n \leq 3$).

B. Communication Analysis for Two-Layer Aggregation System with k-out-of-n Secret Sharing

In our fault-tolerant extension of the two-layer SAC (Sec. IV-C), more shares for peer models are exchanged on the network. In case of k-out-of-n secret sharing, the cost for the first exchange of shares (lines 3–9 in Alg. 4) is $n(n-1)(n-k+1)|w|$ and the one for exchanging subtotals (lines 14–16 in Alg. 4) is $(k-1)|w|$. The communication cost of SAC per round is $\{n(n-1)(n-k+1)+(k-1)\}|w|$, so the cost for m subgroups is $m\{n(n-1)(n-k+1)+(k-1)\}|w|$. Same as Sec. VII-A, costs for FedAvg and broadcasting the weight average from the FedAvg group to all SAC peers are $2(m-1)|w|$ and $m(n-1)|w|$, respectively. Therefore, the total communication cost per aggregation is

$$\{(n^2 - kn + k)N + km - 2\}|w|. \tag{5}$$

Thus, there is a trade-off between redundancy and communication cost. While the communication cost for SAC (k-out-of-n) is greater than that of the n-out-of-n case (Sec. VII-A), the entire cost is still much smaller than the one-layer SAC thanks to the two-layer approach.

Fig. 14 compares the communication costs for different combinations of k and n. Compared to the baseline (green),

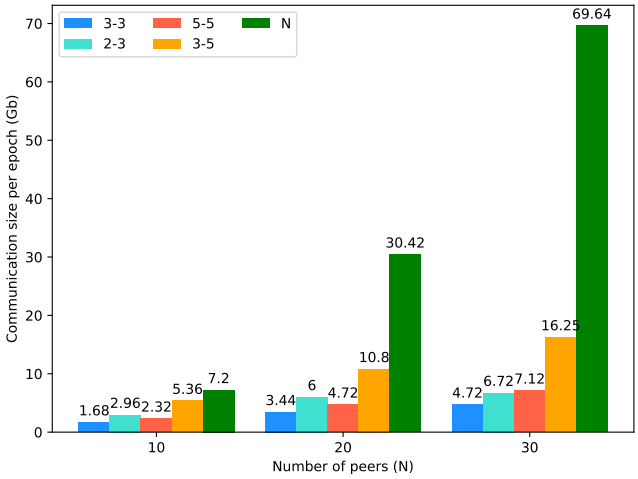

Fig. 14. Total communication cost per aggregation under various k-n settings. We use the CNN model as described in Fig. 5. Each subgroup consists of $n = 3, 5, N$ peers, and is operational as long as k peers are alive. k-n refers to our proposed two-layer SAC with k-out-of-n secret sharing scheme; in the 3-5 setting, the system consists of five peers per subgroup, and each subgroup can perform the aggregation as long as $k = 3$ peers are alive. The $n = N$ case is for the original one-layer aggregation (Alg. 2).

the proposed system offers greater communication efficiency with an increasing N. For instance, it is 8.84x and 14.75x more efficient in case of $n, k, N = 3, 3, 20$ and $3, 3, 30$. While the fault-tolerant version (e.g., $n, k = (3, 2), (5, 3)$) requires more communication, the total cost is still smaller than the baseline. For instance, it is still 10.36x and 4.29x more efficient in case of $n, k, N = 3, 2, 30$ and $n, k, N = 5, 3, 30$. Our method can reduce the communication cost by a factor of 20 with a larger number of peers (especially for $N = 50$) although it is omitted in this figure. The aggregation cost is 196.13Gb in the baseline ($n = N = 50$), which is brought down to 8.24Gb in our method ($n, k, N = 3, 3, 50$), leading to a 23.80x improvement. Considering that our method achieves accuracy on par with the baseline as seen in Fig. 6, our method greatly improves the communication efficiency. Our system also allows flexible choices of both k and n to strike the best balance between communication efficiency and fault tolerance.

C. Communication Analysis for Multi-Layer Aggregation System

Here, we analyze the cost of multi-layer aggregation in the n-out-of-n case. For simplicity, the follower in an x-th layer subgroup becomes a leader in the $x+1$-th layer, but cannot become a leader in the $x+2$-th layer except that the leader of the topmost layer serves as the one of the second layer as well. Note that we assume each subgroup has the same number of peers n and SAC is used across all the layers. The total number of peers N in this X-layer aggregation system is

$$N = \sum_{k=1}^{X} n(n-1)^{k-1}. \tag{6}$$

The communication cost for each aggregation is $(n^2-1)|w|$ and the number of aggregations in all layers is $\sum_{k=1}^{X-1} n(n-1)^{k-1}+1$. The communication cost for distributing aggregated

models is also $(N-1)|w|$, so the total communication cost C_{total} is derived as

$$C_{total} = (n^2-1)\{\sum_{k=1}^{X-1} n(n-1)^{k-1}+1\}|w| + (N-1)|w| \quad (7)$$

$$= (n+1)\{n\sum_{k=1}^{X-1}(n-1)^k + (n-1)\}|w| + (N-1)|w| \quad (8)$$

$$= [(n+1)\{(N-n)+(n-1)\}+(N-1)]|w| \quad (9)$$

$$= (N-1)(n+2)|w|. \quad (10)$$

Therefore, even if SAC is used in all layers including the topmost layer, the communication complexity is still $O(nN)$. When n becomes smaller as the number of layers increases, the communication complexity approaches to $O(N)$. The communication complexity will be further reduced if other aggregation methods with less communication like FedAvg are used instead of SAC.

D. Fault Tolerance Analysis for Two-Layer Raft

In this section, we briefly analyze the security threshold for two-layer Raft consensus in comparison with the original one. In the original Raft, the cluster is available when the majority of servers is running, so FedAvg layer and each subgroup in SAC layer tolerate up to $\lfloor \frac{n-1}{2} \rfloor$ and $\lfloor \frac{m-1}{2} \rfloor$ peers' crashes respectively. Our system tolerates at most $m(\lfloor \frac{n-1}{2} \rfloor + 1)$ faulty peers in the optimistic case where all leaders in subgroups are operational and only the followers may crash. On the other hand, our system cannot operate if $\lfloor \frac{m-1}{2} \rfloor$ subgroup leaders (i.e., members of the FedAvg layer) crash at the same time. In our system, when the subgroup leader crashes, a new subgroup leader is added to FedAvg layer as a follower as described in Sec. V-A1. In this case, the number of peers in FedAvg layer is increased by one through cluster membership change protocol in Raft. The quorum for FedAvg layer increases as well.

VIII. Conclusion

In this paper, we proposed a scalable, secure and fault tolerant aggregation system for P2P FL. By dividing the system into a two-layer network and leveraging the benefits of SAC and FedAvg, our two-layer aggregation system was able to both provide privacy protection of peers and reduce the total communication cost per aggregation. We also proposed a fault tolerant version of the two-layer SAC to handle the random peer dropouts during aggregation. As a backend of the two-layer aggregation system, our two-layer Raft successfully enhances fault tolerance in case of the aggregation leader crashes. Experimental results demonstrate that our proposed system outperforms SAC in terms of the communication cost and fault tolerance with the comparable accuracy. For example, with 30 peers, we have achieved a 10.36x communication cost reduction. We also showed that slow subgroups do not affect the overall accuracy. Our two-layer Raft maintained availability by quickly detecting a crashed leader and replacing it with a new one in various cases. Finally, we discussed aggregation costs when the system is scaled to multiple layers, and the minimum number of peers to maintain consensus in the two-layer Raft.

Acknowledgments This work was partially supported by JST AIP Acceleration Research JPMJCR23U3 and JSPS KAKENHI Grant Numbers JP22H03596, Japan.

References

[1] B. McMahan, E. Moore, D. Ramage *et al.*, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017, pp. 1273–1282.
[2] M. Nasr, R. Shokri *et al.*, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *Proceedings of the IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 739–753.
[3] A. G. Roy, S. Siddiqui *et al.*, "BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning," ArXiv Preprint arXiv:1905.06731, 2019.
[4] T. Wink and Z. Nochta, "An Approach for Peer-to-Peer Federated Learning," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2021, pp. 150–157.
[5] M. R. Behera, S. Shetty *et al.*, "Federated Learning using Peer-to-peer Network for Decentralized Orchestration of Model Weights," TechrXiv Preprint techrxiv.14267468.v1, 2021.
[6] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, 2014, pp. 305–319.
[7] M. Ito, A. Saito *et al.*, "Secret Sharing Scheme Realizing General Access Structure," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 72, no. 9, pp. 56–64, 1989.
[8] K. Bonawitz, V. Ivanov *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (ACM CCS)*, 2017, pp. 1175–1191.
[9] J. So, B. Güler *et al.*, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 479–489, 2021.
[10] W. Li, C. Feng *et al.*, "A scalable multi-layer pbft consensus for blockchain," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 32, no. 5, pp. 1146–1160, 2020.
[11] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 99, no. 1999, 1999, pp. 173–186.
[12] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized business review*, 2008.
[13] D. Evans, V. Kolesnikov *et al.*, "A Pragmatic Introduction to Secure Multi-Party Computation," *Foundations and Trends in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.
[14] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.
[15] L. Lamport, "Paxos made simple," 2001, pp. 51–58.
[16] C. Dwork, "Differential privacy," in *Proceedings of the International Colloquium on Automata, Languages and Programming, part II (ICALP)*, vol. 4052, 2006, pp. 1–12.
[17] HashiCorp, "raft (github repository)," https://github.com/hashicorp/raft, 2018.
[18] Google, "gRPC," https://grpc.io/, 2024.
[19] Y. LeCun, C. Cortes *et al.*, "The MNIST Database of Handwritten Digits," http://yann.lecun.com/exdb/mnist/, 2010.
[20] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," 2009.
[21] B. Hubert, "tc(8) — Linux manual page," https://man7.org/linux/man-pages/man8/tc.8.html, 2001.
[22] H. Howard, M. Schwarzkopf *et al.*, "Raft refloated: Do we have consensus?" *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 12–21, 2015.