

Towards Practical Overlay Networks for Decentralized Federated Learning

Yifan Hua¹, Jinlong Pang¹, Xiaoxue Zhang¹, Yi Liu¹, Xiaofeng Shi¹, Bao Wang², Yang Liu¹, Chen Qian¹

¹University of California Santa Cruz, ²University of Utah

{yhua294, pang14, xzhan330, yliu634, xshi24, yangliu, cqian12}@ucsc.edu bwang@math.utah.edu

Abstract—Decentralized federated learning (DFL) uses peer-to-peer communication to avoid the single point of failure problem in federated learning and has been considered an attractive solution for machine learning tasks on distributed devices. We provide the first solution to a fundamental network problem of DFL: what overlay network should DFL use to achieve fast training of highly accurate models, low communication, and decentralized construction and maintenance? Overlay topologies of DFL have been investigated, but no existing DFL topology includes decentralized protocols for network construction and topology maintenance. Without these protocols, DFL cannot run in practice. This work presents an overlay network, called FedLay, which provides fast training and low communication cost for practical DFL. FedLay is the first solution for constructing near-random regular topologies in a decentralized manner and maintaining the topologies under node joins and failures. Experiments based on prototype implementation and simulations show that FedLay achieves the fastest model convergence and highest accuracy on real datasets compared to existing DFL solutions while incurring small communication costs and being resilient to node joins and failures.

I. INTRODUCTION

Training machine learning (ML) models using data collected by distributed devices, such as mobile and IoT devices, is crucial for modern ML. Federated learning (FL) [1], [2], [3], [4], [5], [6], [7] has become a popular ML paradigm that allows a large number of clients (end systems, edge nodes, etc.) to train ML models collaboratively without directly sharing training data. FL uses a central server or cloud to orchestrate clients for training ML models and iterates the following procedure: The server creates a global model by aggregating the local ML models collected from the clients and then sends it to clients for edge applications; the ML models are updated at the clients. Compared to collecting raw data from distributed devices and performing centralized ML, FL has several main advantages, including saving communication costs on limited-bandwidth devices, preserving data privacy, and being compatible with country or organization regulations that prohibit direct data sharing.

However, the drawbacks of FL are also prominent and have been studied and widely mentioned in the literature [3], [8], [9]. For example, the central orchestration server that frequently exchanges models with clients clearly presents a bottleneck and becomes a typical *single point of failure* [3], [10]. In addition, the server is also a single point of attack: adversaries can make all clients use tampered ML models by attacking the server. There is even a risk that the server itself

is malicious, which might distribute incorrect global models or collect sensitive information from the clients.

Decentralized federated learning (DFL) emerged recently [8], [17], [10] to resolve the above problems of FL, by removing the involvement of the central server. DFL clients form a peer-to-peer (P2P) network and keep exchanging their models using P2P communication. In most cases, the data on different clients are not identically and independently distributed (non-iid) [8], [17], [10], hence the trained local ML models are substantially different from each other. After sufficient model exchanges, the local models on the clients may converge to a model that correctly reflects the features of data from all clients.

This work focuses on a *fundamental network problem* of DFL: what overlay network is ideal for DFL in practice? An overlay network of DFL is a logical network on top of the physical networks. It specifies which pairs of clients should exchange their local models: two clients exchange models if they are overlay neighbors. An ideal overlay network for DFL needs to satisfy a few requirements [10] including 1) a decentralized construction protocol that can build the overlay topology; 2) fast convergence of local models to high accuracy; 3) small node degree that can maintain low bandwidth cost on clients for exchanging models with a limited number of neighbors; 4) resilient to client dynamics such as client joins, leaves, and failures – they are also called as *churn*.

Table I shows a list of overlay network topologies that have been studied for DFL. We find that most of these existing studies do not pay attention to whether the proposed topologies can be constructed by decentralized protocols and resilient to churn. These two requirements are common networking/distributed system problems and might not be the focus for ML researchers.

DFL cannot work as a practical system without a decentralized construction protocol for its overlay network.

For example, recent work suggests that Ramanujan graphs provide fast convergence and accurate models for DFL [18], [16]. However, the decentralized construction of Ramanujan graphs is unknown. Centralized construction/maintenance contradicts the main purpose of DFL: avoid the single point of failure/attack.

We propose a fully decentralized overlay network for DFL, called FedLay, which achieves all four requirements discussed above, namely decentralized construction, fast convergence to accurate models, small communication cost, and resilience to

| Overlay network | Decentralized construction | Node degree | Model convergence | Resilience to churn | Other comments |
|------------------------------|----------------------------|-------------|-------------------|---------------------|--|
| Ring [8] | Not discussed | 2 | Slow | Not discussed | |
| 2D grid [8] | Not discussed | 4 | Slow | Not discussed | |
| Complete graph [8] | Not discussed | $N-1$ | Fast | Not discussed | |
| Dynamic chain [11] | Not discussed | 2 | Faster than ring | Not discussed | |
| D-Cliques [12] | Unknown | $ C -1$ | Fast | Not discussed | Assume global knowledge |
| Clustering [13] | Not discussed | $ C -1$ | Fast | Not discussed | Bottlenecks exist |
| Hypercube [14] | Not discussed | $O(\log N)$ | Fast | Not discussed | |
| Torus [14] | Not discussed | d | Fast | Not discussed | |
| Ramanujan [15], [16] | Not discussed | d | Fast | Not discussed | |
| Random d -graph [15], [16] | Unknown | d | Fast | Not discussed | Assume global knowledge |
| FedLay (this work) | Yes | d | Fast | Yes | Address device/data heterogeneity |

TABLE I: List of overlay network topologies for DFL. N is the number of clients. d is a small constant for node degree, usually around 10. $|C|$ is the size of each cluster/cliue, usually bigger than values of d . Other than FedLay, only D-Cliques and [16] discuss its construction algorithm but still assumes global knowledge.

churn. FedLay does not need a centralized server at any stage and all clients run the same suite of distributed protocols. The FedLay protocol suite includes two sets of protocols: 1) a set of *Neighbor Discovery and Maintenance Protocols (NDMP)* to build the overlay network and recover it from churn; and 2) a *Model Exchange Protocol (MEP)* to achieve fast model convergence for heterogeneous clients and asynchronous communication. The FedLay topology is motivated by the near-random regular topologies that have been proposed for data center networks [19], [20]. **However, [19] [20] are centralized protocols for data centers and cannot be applied to DFL.** To our knowledge, FedLay is the first solution for constructing near-random regular topologies in a decentralized manner and maintaining the topologies under node joins and failures. FedLay also considers other practical issues, including non-iid data and asynchronous communication with heterogeneous clients.

The contributions of this work are summarized as follows.

- We identify three topology metrics related to DFL convergence and evaluate various overlay topologies of DFL. We find that FedLay outperforms all other topologies. FedLay, as a decentralized network, has almost identical results on all three metrics to the best result among the 100 randomly generated regular graphs (in a centralized way).
- We design and implement the FedLay protocol suite. **To our knowledge, FedLay is the first DFL overlay network that provides decentralized protocols for construction, churn recovery, and model aggregation.**
- We evaluate FedLay using both prototype implementation and simulations on real ML datasets. We find that FedLay achieves the highest average model accuracy and fastest convergence compared to other DFL methods. It also has small communication cost and strong resilience to churn.

The rest of this paper is organized as follows. Section II presents the metrics for selecting DFL overlay topologies and the details of the topology of FedLay. The design of FedLay protocol suite, including the topology construction and maintenance protocols and model aggregation protocol, is presented in Section III. Section IV shows the evaluation results of FedLay as well as existing DFL overlay networks on real ML datasets. We present related work in Section V

and conclude this work in Section VI.

II. OVERLAY TOPOLOGY OF FEDLAY

This section presents the topology of FedLay and the intuition behind using this topology. We first explore what topology metrics can be used to evaluate the convergence speed under small node degrees. Then we design the FedLay topology and use numerical results to show its advantages. The decentralized construction and maintenance under churn will be presented in the next section.

A. Three metrics for DFL topologies

A DFL topology can be modeled as an undirected graph $G = (V, E)$, where each node $v \in V$ represents a client in the DFL system and each link $e = (u, v) \in E$ indicates that two clients u and v will exchange local ML models – u and v are thus called neighbors. We assume clients have equal roles in the overlay and similar numbers of neighbors.

1) *Expander property and convergence factor.*: An important notion in DFL, or general decentralized optimization algorithms, is the mixing matrix \mathbf{M} of the graph G . The i -th row of \mathbf{M} denotes the weights used for aggregating local models of the neighboring nodes to update the model of the i -th client. Hence the adjacency matrix of an overlay network and its Metropolis-Hastings matrix are both mixing matrices [21]. The symmetric property of \mathbf{M} indicates that its eigenvalues are real and can be sorted in non-increasing order. Let $\lambda_i(\mathbf{M})$ denote the i -th largest eigenvalue of \mathbf{M} , then we have $\lambda_1(\mathbf{M}) = 1 > \lambda_2(\mathbf{M}) \geq \dots \geq \lambda_N(\mathbf{M}) > -1$ based on the spectral property of the mixing matrix [21]. The constant $\lambda = \lambda(\mathbf{M}) := \max\{|\lambda_2(\mathbf{M})|, |\lambda_N(\mathbf{M})|\}$ has been used to characterize optimization error (a measure of training loss) and generalization gap (a measure of test accuracy) of DFL. In particular, it is shown that the optimization error and generalization gap – for a typical DFL framework, Decentralized Federated Averaging (DFedAvg) – are bounded, respectively, by $\mathcal{O}\left(\frac{1}{(1-\lambda)^2}\right)$ and $\mathcal{O}\left(2\lambda^2 + 4\lambda^2 \ln \frac{1}{\lambda} + 2\lambda + \frac{2}{\ln \frac{1}{\lambda}}\right)$ – in terms of λ [17], [16]. Notice that both $\frac{1}{(1-\lambda)^2}$ and $2\lambda^2 + 4\lambda^2 \ln \frac{1}{\lambda} + 2\lambda + \frac{2}{\ln \frac{1}{\lambda}}$ are increasing functions of $\lambda \in (0, 1)$.

Per the above discussion, to achieve good convergence and generalization, a topology needs to have a λ sufficiently

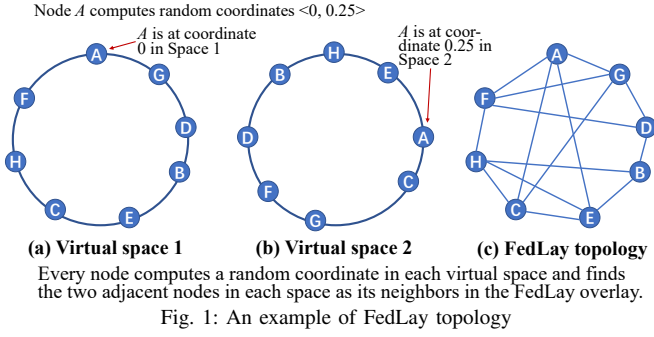


Fig. 1: An example of FedLay topology

smaller than 1 and hence achieve a small value of $\frac{1}{(1-\lambda)^2}$ and $2\lambda^2 + 4\lambda^2 \ln \frac{1}{\lambda} + 2\lambda + \frac{2}{\ln \frac{1}{\lambda}}$. Thus we define the first topology metric, called the *convergence factor* of G : $c_G = \frac{1}{(1-\lambda)^2}$.

Note that when $\frac{1}{(1-\lambda)^2}$ is minimized, $2\lambda^2 + 4\lambda^2 \ln \frac{1}{\lambda} + 2\lambda + \frac{2}{\ln \frac{1}{\lambda}}$ is also minimized. Hence for the sake of simplicity, we do not need another factor.

2) *Network diameter*.: The diameter of a network is the longest length of all shortest paths calculated in the network. It reflects the network distance between the two most distant nodes. The intuition of considering this metric is that the network diameter can represent the maximum latency that the local model trained on the data of a client can propagate to all clients in the network.

3) *Average length of shortest paths*.: The third metric is the average length of all shortest paths in the network. The intuition of considering this metric is that the average length can represent the average latency that a local model can propagate to a random client.

B. FedLay topology

Recent theoretical studies show that Ramanujan graphs can provide small values of the spectral expander property λ and hence achieve ‘optimal’ convergence with a constant node degree d [16]. However, a large Ramanujan graph cannot be generated even by centralized construction. Hence, random regular graphs (RRGs) can be used instead, which are approximately Ramanujan for a large network size n [15]. Note that RRGs cannot be generated with any deterministic algorithm either. Hence, near-RRGs are usually used in practice, which are considered close enough to RRGs [19], [20]. near-RRGs can achieve ideal values on both the convergence factor and shortest path lengths.

A practical problem is that all existing near-RRGs are constructed by centralized methods [19], [20], [22], [16]. FedLay is the first attempt to achieve a near-RRG using decentralized construction. In FedLay, each node computes a set of *virtual* coordinates C , which is an L -dimensional vector $\langle x_1, x_2, \dots, x_L \rangle$ where each element x_l is a random real number in range $[0, 1]$. In practice, x_i can be computed as $H(IP_x | i)$ where H is a publicly known hash function and IP_x is x ’s IP address.

We define L virtual ring spaces. In the i -th ring space, a node is *virtually* placed on a ring based on the value of its i -th coordinate x_i . The coordinates of each space are circular, with 0 and 1 being superposed at the top-most point of the

ring and 0.5 being the bottom-most point. If the coordinates of two different nodes are identical in one space, their orders on the ring are determined by the values of their IP addresses. For ease of presentation, we assume all coordinates on a ring are different. As shown in the example in Fig. 1, there are 8 nodes and each of them computes a set of two-dimensional *random* coordinates $\langle x_1, x_2 \rangle$. There are two virtual ring space as shown in Figs. 1 (a) and (b) and every node is on a position of the i -th ring based on its random coordinate x_i . Note, all spaces are virtual and they have no relationship to the geographic locations of the nodes.

In each virtual space, every node u has two adjacent nodes on the ring, based on the order of their coordinate values. u will find the adjacent nodes from all spaces as its overlay neighbors (by a decentralized protocol described later) for model exchange. In the example of Fig. 1(c), every node finds its adjacent nodes in two spaces and forms the FedLay overlay. So most nodes have four neighbors in the overlay but there are a few ones, like node B , has only three neighbors because D is adjacent to B on both rings. Hence, for L spaces, every node has at most $2L$ neighbors. L can then be considered as a parameter for communication and convergence trade-off: with a bigger L , nodes have more neighbors for model exchanges but increased communication cost.

The detailed decentralized construction and maintenance will be discussed in the next section. We now show that the FedLay topology is close to the optimal choice with a given node degree d , measured by the three metrics discussed in Section II-A. We evaluate the FedLay topology by comparing it with the following existing topologies.

- 1) **Best of 100 randomly generated d -regular graphs (“Best”)**. We randomly generate 100 d -regular graphs and measure them for the three metrics. We obtain the optimal value among the 100 graphs for each metric. It is then considered the optimal value of practical topologies.
- 2) **Chord [23]**. Chord is a well-known peer-to-peer overlay network serving the function of a distributed hash table (DHT). It has a $O(\log n)$ degree and can be constructed and maintained by decentralized protocols.
- 3) **Viceroy [24]**. Viceroy is a peer-to-peer overlay network with a constant degree, inspired by the classic Butterfly network used for super-computing. Its main objective is to minimize congestion by overlay routing. It can be constructed and maintained by decentralized protocols.
- 4) **Distributed Delaunay Triangulation (DT) [25], [26]**. DT is an overlay network with a constant degree that supports greedy routing. It can be constructed and maintained by decentralized protocols.
- 5) **Waxman network [27]**. Waxman is a network that simulates connections with physical proximity. Nodes with a close geographic distance are likely to connect. There is no known decentralized construction of Waxman.
- 6) **Social network [28]**. We use a social network topology of Facebook users that was collected by [28]. This is a typical example of overlay networks that rely on information from other application channels.

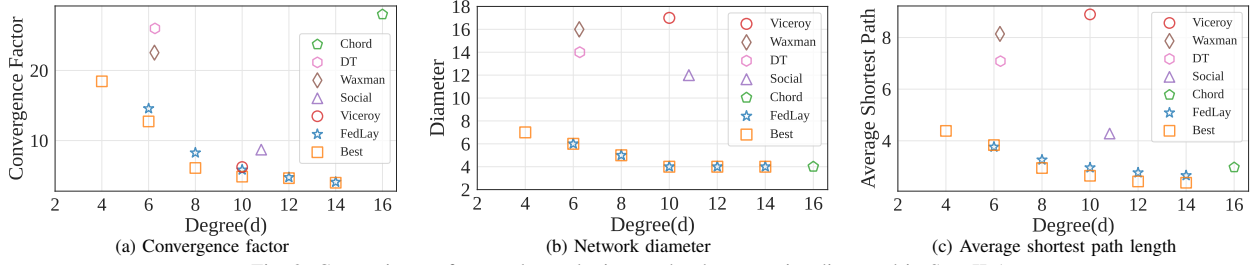


Fig. 2: Comparisons of network topologies on the three metrics discussed in Sec. II-A.

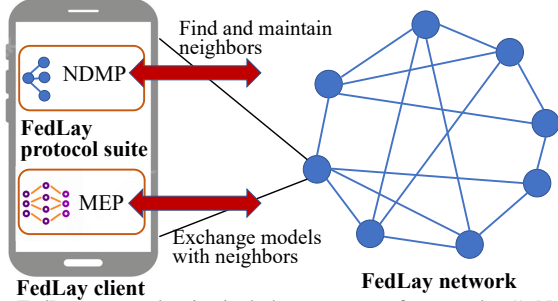


Fig. 3: FedLay protocol suite includes two sets of protocols: 1) Neighbor Discovery and Maintenance Protocols; 2) Model Exchange Protocol.

Fig. 2 shows the empirical results by comparing all the above-mentioned topologies on the three metrics: the convergence factor, diameter, and average length of shortest paths. For all metrics, smaller values are more desired. Each network includes 300 nodes for fair comparisons. We vary the node degree from 4 to 14 for both “Best” and FedLay. Other networks do not support flexible node degrees, so the result of each topology is shown as a single dot in each figure.

We summarize our findings from these results as follows. “Best” always provides optimal results for every metric. The results of FedLay are extremely close to “Best”: most points of FedLay are superposed with those of “Best” with only a few exceptions. All of topologies are much less optimal compared to FedLay. The convergence factor of Chord is very high but the diameter and average shortest path length are low, due to its high node degree. **FedLay achieves the best results on all three metrics among existing practical overlay topologies.**

Definition 1 (A correct FedLay overlay). *We define that a FedLay network is correct, if every node u has a neighbor set N_u such that N_u includes the adjacent nodes of u in all L virtual ring spaces and does not include other nodes. Each node also knows the virtual coordinates of all its neighbors.*

III. DESIGN OF FEDLAY PROTOCOLS

A. Overview

As shown in Fig. 3, the FedLay protocol suite is completely decentralized, running on each client and consisting of two sets: 1) Neighbor Discovery and Maintenance Protocols (NDMP); and 2) Model Exchange Protocol (MEP). The objective of NDMP running on node u is to allow u to find its correct neighbors during the join of u and maintain correct neighbors under network dynamics. Hence, NDMP can be considered as *control protocols* to construct the correct FedLay network. The objective of MEP is to decide when

to exchange the local models with the neighbors and how to process the received models. Hence, MEP can be considered as an *application protocol* to optimize the model convergence of DFL. Both NDMP and MEP use TCP with reliable delivery.

B. Neighbor Discovery and Maintenance Protocols (NDMP)

NDMP includes join, leave, and maintenance protocols. The join protocol is run by each new node joining the FedLay network. It ensures all nodes will find the correct neighbors after the node joins the network. The leave protocol is run by each node that is preparing to leave the network. It ensures all remaining nodes will keep the correct neighbors after the leave. The maintenance protocol is run by every node periodically to detect potential failed neighbors or wrong neighbors and fix these errors.

1) *NDMP join protocol*: The join protocol is designed to achieve the following correctness property: given an existing FedLay overlay network, a new joining node runs the join protocol. When the join protocol finishes, the joining node is guaranteed to find its correct neighbors, i.e., the adjacent nodes in all virtual spaces. Note that such recursive property is a key module to ensure the correctness of many P2P overlays such as Chord [23] and distributed DT graph [25], [26].

FedLay builds upon the concept of random virtual coordinates and circular distance introduced by SpaceShuffle [20], a centralized solution. This work designs a fully decentralized topology construction for FedLay. FedLay allows any client to join the network through any existing node without an administrator, enhancing scalability and user flexibility. Additionally, we optimize the NDMP leave protocol to minimize overhead by ensuring that maintenance operations are only triggered when necessary, thus reducing resource consumption during network changes.

When a new node u joins the existing correct FedLay network, we assume u knows one existing node v in the overlay, which can be an arbitrary node – this is the minimum assumption for any overlay network. If u knows no existing node, it has no way to join any overlay. u first computes a random coordinate as its position in the first virtual space, say x_1^u . u will let v sends a message *Neighbor_discovery* to the current FedLay network using *greedy routing* to the destination location x_1^u . *Neighbor_discovery* also includes u ’s IP address.

We first define the concept of *circular distance*, which is a metric used in greedy routing of FedLay.

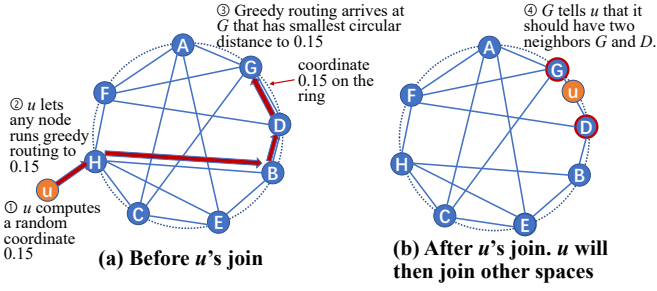


Fig. 4: An example of the FedLay join protocol.

Definition 2 (Circular distance). *The circular distance of two coordinates x and y in the same ring space, $0 \leq x, y < 1$, is:*

$$CD(x, y) = \min\{|x - y|, 1 - |x - y|\}.$$

For two coordinates x and y on a ring, the circular distance is the length of the smaller arc between x and y , normalized by the perimeter of the ring that is 1. We say x is closer to y than w on a ring space, if $CD(x, y) < CD(w, y)$. If x and w have the same circular distance to y , we always break the tie to one of x and w with a smaller value of their IP addresses (considering each IP address is a 32-bit value). Hence, there is only one node that is closest to a given coordinate x .

Upon receiving *Neighbor_discovery*, the greedy routing protocol to the destination location x_i^u in Space i is executed by a node v as following:

- 1) Node v finds a neighbor w , such as w 's coordinate in Space i , x_i^w , has the smallest circular distance to x_i^u among all neighbors of v .
- 2) If $CD(x_i^u, x_i^v) > CD(x_i^u, x_i^w)$, v forwards the *Neighbor_discovery* message to w .
- 3) If $CD(x_i^u, x_i^v) \leq CD(x_i^u, x_i^w)$, *Neighbor_discovery* stops at v . From v 's two adjacent nodes, v finds the adjacent node p such that x_i^u is on \widehat{vp} , the smaller arc between v and p . Then v sends a message to u to tell u that v and p are u 's adjacent nodes on this virtual ring and let u add v and p to u 's neighbor set.

The greedy routing presented above will make each node forward *Neighbor_discovery* to its neighbor that has the shortest circular distance to the destination location x_i^u . When a node v cannot find a neighbor that is closer to x_i^u than v itself, v must be the node that has the shortest circular distance to x_i^u among all nodes in FedLay (will be formally proved later). Hence, v and one of its adjacent nodes p will be u 's neighbors.

We show an example of FedLay join in Fig. 4. u joins FedLay and it knows an existing node H in the network. u computes a random coordinate 0.15 and asks H to run greedy routing of *Neighbor_discovery* to 0.15. H will forward *Neighbor_discovery* to B , which is closest to 0.15 in the space among all H 's neighbors. Eventually, the message arrives at G , the node that is closest to 0.15 in the space and G tells u to add G and D as neighbors. Note greedy routing has much smaller hop-count than traveling nodes one after another through the ring because there are many shortcuts like the link HB .

We have the following property.

Lemma 1. *In a ring space of a correct FedLay network and*

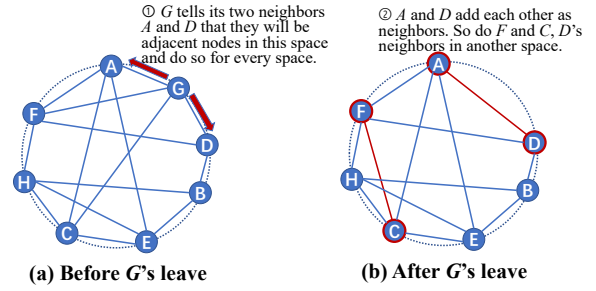


Fig. 5: An example of the FedLay leave protocol.

a given coordinate x , if a node v is not the node that has the smallest circular distance to x in the space, then v must have an adjacent node w on the ring such that $CD(x, x^v) > CD(x, x^w)$, where x^v is v 's coordinate.

The proof can be found in [29].

Note that every adjacent node of v is its neighbor in a FedLay network. This lemma tells that if v is not the node that is closest to the destination coordinate x_i^u , then the greedy routing algorithm must execute Step 2 and forwards the message to a neighbor. So we have,

Theorem 1. *In a FedLay network, when *Neighbor_discovery* to the destination coordinate x stops at a node v , v must be the node that has the smallest circular distance to x .*

The *Neighbor_discovery* message will stop at the node v closest to x_i^u and v must be an adjacent node to the joining node u , because no other node is closer to u 's coordinate x_i^u . v also knows the other adjacent node w of u , because w is a current adjacent node of v . v will send a message to u by TCP including the information of v and w . u will then add v and w to its neighbor set.

The joining node u can find all its neighbors by running the above join protocol in all spaces. Therefore, **if u joins a correct FedLay network, the new FedLay network after this join is also correct.**

In some extreme cases, there could be multiple nodes joining the network simultaneously. This situation will be handled by both the join and maintenance protocols.

2) *NDMP leave protocol*: The leave protocol of NDMP is quite straightforward. When a user wants to leave and closes the client program of FedLay, for every virtual space, the leaving node sends messages to its two adjacent nodes and tells them to add each other to their neighbor sets. As shown in Fig. 5, node G wants to leave the network and tells its two adjacent nodes A and D about its leaving. Then A and D will consider each other as adjacent nodes and neighbors. In another space, G 's two adjacent nodes F and C will also add each other to their neighbor sets. Hence, **the new FedLay network after a node leave is also correct.**

3) *NDMP maintenance protocol*: An overlay network may experience node failures due to various reasons such as an Internet service outage and end system failures. A failed node disappears without notice. In order to detect and fix these situations, each node in FedLay also runs the maintenance protocol.

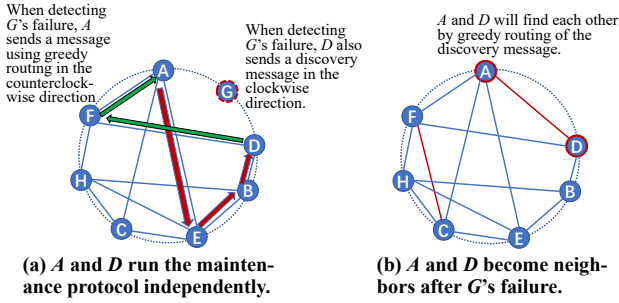


Fig. 6: An example of the maintenance protocol.

The maintenance protocol requires every node to send each of its neighbors a heartbeat message periodically. Suppose the time period between two heartbeat messages is T . If a node p has not received any heartbeat message from a neighbor u for $3T$ time, it considers that u has failed. p then sends a *Neighbor_repair* message by greedy routing in the opposite direction of u on the virtual space i where u and p are adjacent nodes. In the example of Fig. 6(a), node A detects the failure of G . Since G is an adjacent node of A on A 's clockwise side, A sends a *Neighbor_repair* message by greedy routing in the counterclockwise direction (the opposite direction of G). By “counterclockwise direction”, it requires that all hops of such greedy routing, $A - E - B - D$ in this example, should follow the counterclockwise order.

We give a formal description of the counterclockwise direction: Upon receiving *Neighbor_repair* to x_i^u in Space i , a node v runs the following algorithm:

- 1) Node v considers a subset of its neighbors, such that for every neighbor w in the subset, w 's coordinate in Space i , x_i^w , satisfies $x_i^w < x_i^v$ or $x_i^w > x_i^u$. From the above subset, node v finds a neighbor w' , such that w' 's coordinate in Space i , $x_i^{w'}$, has the smallest arc length $L(x_i^{w'}, x_i^u)$ among all neighbors in the subset.
- 2) If $L(x_i^v, x_i^u) > L(x_i^{w'}, x_i^u)$, v forwards *Neighbor_repair* to w' .
- 3) If $L(x_i^v, x_i^u) < L(x_i^{w'}, x_i^u)$, *Neighbor_repair* stops at v . v then tells p that v is p 's adjacent nodes on this virtual ring and let p add v to p 's neighbor set.

When *Neighbor_repair* stops, it arrives at another adjacent node of G before G 's failure (stated in a later theorem). Then G 's previous two adjacent nodes can be connected.

In Fig. 6(a), node D also detects G 's failure, independent of A 's detection. Then D sends a *Neighbor_repair* message to the clockwise direction and the message will travel on a path $D - F - A$. The algorithm to forward *Neighbor_repair* in the clockwise direction can be specified in a similar way.

Theorem 2. Consider a correct FedLay network and a node u fails. When a node p detects the failure of its adjacent node u in Space i , it sends *Neighbor_repair* to the destination coordinate x_i^u in the opposite direction of u . When *Neighbor_repair* stops at a node q , q is another adjacent node of u in Space i before u 's failure.

The proof can be found in [29].

Based on the theorem, in every virtual space, the two

adjacent nodes of the failed u can find and connect each other. Hence, if a node fails in a correct FedLay network, after the node failure FedLay is still correct.

Neighbor repair for concurrent joins and failures. Note the above property cannot be proved for multiple failures that happen at the same time, called concurrent failures. For concurrent joins and failures, we allow each node u to periodically send two *Neighbor_repair* messages with destination x_i^u to both counterclockwise and clockwise directions in every virtual space i , even without detecting any neighbor failure. For each node v in u 's neighbor set, if they are indeed adjacent in a virtual space, v will receive a *Neighbor_repair* message that stops at v . The *Neighbor_repair* stops at a node w that is not in u 's neighbor set, w and u will add each other to their neighbor sets. We conduct experiments of extreme concurrent joins and failures and the above method always allows the network to recover to a correct FedLay.

C. Model Exchange Protocol (MEP)

One key challenge of DFL is that there is no central server to evaluate the quality of models from different clients. In P2P model exchanges, a client with low-quality local models can ‘infect’ its neighbors with high-quality models and these errors may be further propagated in the overlay. MEP is designed to limit the impact of low-quality models and amplify the impact of high-quality models in a decentralized way. We consider two practical issues in DFL systems. 1) Data heterogeneity [8], [10], [16]. It is well known that the local data of different clients are usually non-iid due to geographic and environmental diversity. Hence, their models have different accuracy. 2) Client heterogeneity [30], [31], [32]. Clients of DFL could have different bandwidth and computing capacities. They may have different model exchange frequencies.

1) Asynchronous model exchange: Previous work assumes synchronous, round-based communication, in which all clients use the same time period to exchange models with neighbors [8], [11], [12], [14], [15], [16]. However, due to client heterogeneity, some low-resource clients may become “stragglers” that will fail to perform model exchanges in the given time period, while powerful or newly joined clients prefer shorter time period (or higher frequency) of model exchanges.

MEP uses asynchronous communication and allows each client u to use a different communication time period T_u . T_u can be set in two ways: **1) Coarse-grained settings.** Each client may configure a period based on their device and communication types, for example, Server-LAN, PC-LAN, Laptop-WLAN, Phone-LTE, and IoT-WLAN. These values are pre-specified in the client program. **2) Fine-grained settings.** Based on the monitoring of available bandwidth and computing resources, client u estimates the minimum time duration $T_{u,\min}$ to produce an updated ML model and transmit it to all neighbors. Then its communication period $T_u = \eta T_{u,\min}$ for constant $\eta > 1$.

For two neighbors with periods T_u and T_v , their model exchange period is set to $\max(T_u, T_v)$. Hence, a client may have different exchange periods to different neighbors.

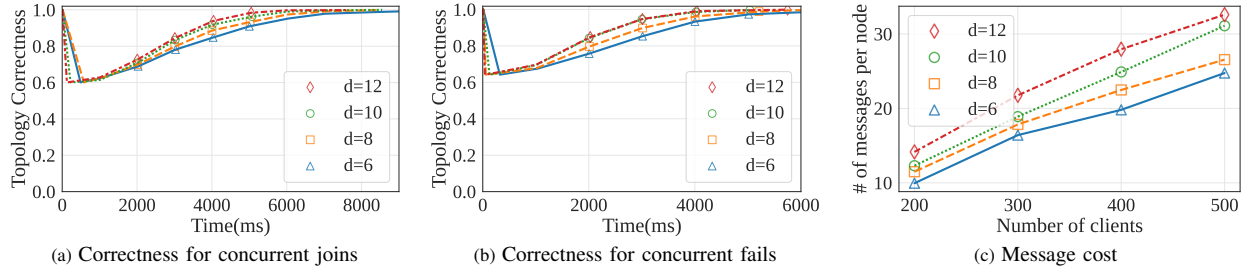


Fig. 7: Topology correctness under churn & message cost

2) *Set confidence parameters*: One key innovation in MEP is to introduce *confidence parameters*. Each node has a set of confidence parameters that present its self-evaluation of the local model accuracy.

We define the data divergence confidence c_d on a client u :

$$c_d^u = \frac{1}{\exp(DKL(D_{loc}||D_{std}))}$$

where $DKL(\cdot)$ is the Kullback-Leibler divergence [33] to evaluate the statistical distance between two probability distributions P and Q , D_{loc} denotes the local data distribution, and D_{std} denotes the estimated iid distribution of the dataset. The uniform distribution is widely used [34], [35] to estimate the iid data because the majority of publicly-available datasets for classification follow uniform distributions, such as MNIST [36] and CIFAR-10 [37]. The Kullback-Leibler divergence can effectively represent the richness of a local dataset. $c_d \in (0, 1]$ and a higher value represents a higher quality of local data and local models.

In addition, we define the communication confidence c_c on a client u : $c_c^u = \frac{1}{T_u}$. The intuition of using c_c is that when a client has more frequent model exchanges with its neighbors, its models are more likely to have higher qualities.

Hence, the overall confidence of client u is

$$c^u = \alpha_d \frac{c_d^u}{\max(c_d)} + \alpha_c \frac{c_c^u}{\max(c_c)}$$

where $\max(c_d)$ and $\max(c_c)$ are the maximum values of c_d and c_c respectively, from all u 's neighbors. α_d and α_c are two constants to balance the weights of the two confidence parameters. The specific values of α_d and α_c can just be 0.5 and 0.5. We try a variety of combinations of α_d and α_c , and in all cases, FedLay achieves fast model convergence on different nodes.

The models from u 's neighbors are aggregated as follows:

$$\omega^u = \frac{\sum_{j \in N \cup \{u\}} c^j \omega^j}{\sum_{j \in N \cup \{u\}} c^j}$$

The above aggregation will be computed once every local time period T_u and the models from each neighbor are always the most updated ones from the neighbor. In this way, clients with low confidence in their model accuracy will have less impact on other clients. In this work, we do not consider the situation where a client might intentionally set a large confidence value to mislead other clients.

IV. PERFORMANCE EVALUATION

A. Evaluation methodology

1) *Three types of evaluation*: We conduct three types of evaluation of FedLay for different scales of DFL networks.

1) **Real experiments**. We conduct experiments with real packet exchanges and data training. We deploy 16 instances to public clouds (we used both Oracle OCI and Amazon EC2), each with a 2GHz CPU and 2GB RAM. Each instance is connected to the Internet and runs a FedLay client. Each client sends and receives NDMP and MEP messages using TCP. Clients train ML models on their local datasets with Pytorch [38] and exchange the models with active neighbors. There is no central server for any purpose, and the system is completely decentralized. The purpose of this type of experiment is to present a prototype and demonstrate that FedLay can run with real ML data training in practice.

2) **Medium-scale emulation with real data training**. In this type of experiment, we use real data training and simulated packet exchanges as discrete events to evaluate networks with up to 100 clients. The simulation and real data training and testing run on a machine with an NVIDIA GeForce RTX3080 graphic card for training acceleration. The purpose of this type of experiment is to evaluate the overlay construction/maintenance, model accuracy, convergence speed, and message cost of FedLay and other DFL methods.

3) **Large-scale simulation with trained models**. For more than 100 clients, conducting all data training on a few machines takes very long time. For networks with 200 to 1000 nodes, we re-use the models trained from the above two types of experiments and assign them to the simulated clients. Packet exchanges are simulated as discrete events.

2) *ML datasets and models*: We evaluate the performance of FedLay for three ML tasks, including 1) Multilayer Perceptron (MLP) for digit classification on the MNIST dataset [39]. 2) Convolutional Neural Networks (CNN) for image classification on the CIFAR-10 dataset [37]. 3) Long Short-Term Memory (LSTM) for role forecasting on the Shakespeare dataset [40] built from *The Complete Works of William Shakespeare*. All three are standard datasets for FL benchmarks [40].

Learning with non-iid data. We generate non-iid MNIST and CIFAR-10 datasets by selecting limited labels for the local training sets using the sharding method. Each shard contains only one label, and each local dataset includes a limited number of shards, resulting in a non-iid distribution and heterogeneity among clients' local datasets.

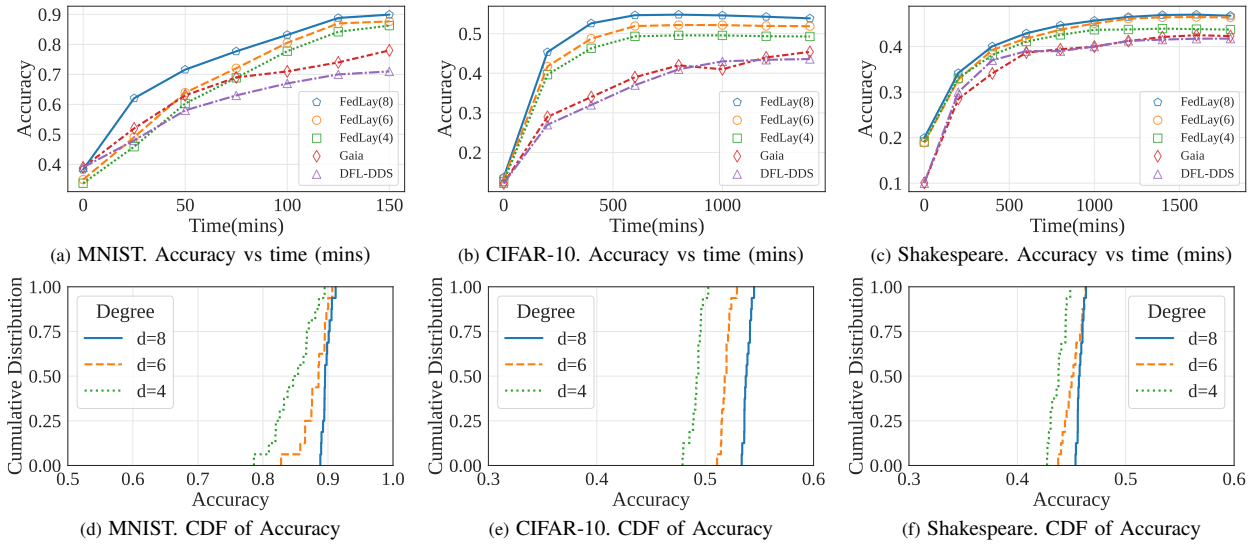


Fig. 8: Model accuracy from 16-client real experiments in Amazon EC2.

Client heterogeneity. We also assumed that clients have different computation and communication resources. We set 3 tiers of clients. For the 16-client real-world experiments, we set 10 medium-capacity, 3 high-capacity, and 3 low-capacity clients. For simulations, each experiment includes 60% medium-, 20% high-, and 20% low-capacity clients. The training time and communication time period of a high-capacity client are $2/3$ of those of a medium-capacity user, and those of a low-capacity client are $2\times$ of those of a medium-capacity user.

3) *Performance metrics:* Besides the topology metrics discussed in Sec. II-A, we use the following metrics to evaluate FedLay and other methods.

Model accuracy: We evaluate the individual accuracy and average accuracy of local ML models based on separate test datasets that are different from the training datasets.

Topology correctness: It is defined as the number of correct neighbors of all nodes over the total number of neighbors. Hence correctness equal to 1 means a correct FedLay.

Communication Cost: We count the number of NDMP messages sent by each client and the total size of the models sent by each client in bytes.

4) *Methods for comparison:* There is no existing DFL topology that allows decentralized construction and maintenance. Hence we compare FedLay with the following methods: 1) Gaia [41] is an ML method for geo-distributed clouds and still uses central servers. Hence it is not DFL. It runs server-based ML in each region and lets servers from different regions connect as a complete graph. It includes no aggregation method to handle non-iid data. 2) DFL-DDS [42] is a DFL method without a fixed topology. Instead, it simulates mobile nodes in a road network and considers two geographically close nodes as neighbors. 3) Chord [23]. 4) FedAvg [1] is a standard centralized FL method. We use its accuracy as the *upper bound* of DFL model accuracy because the central server knows all models from the clients.

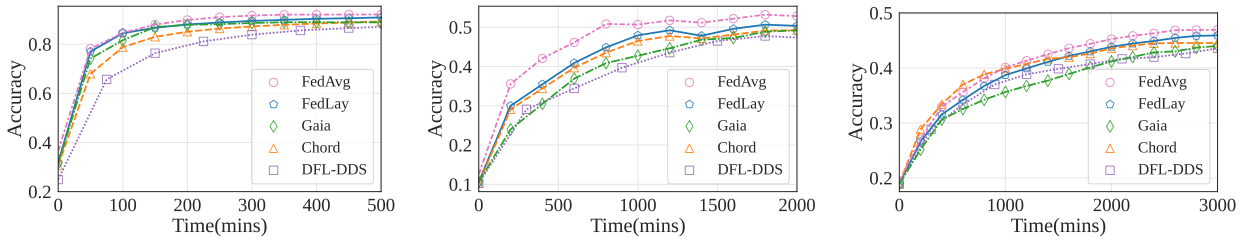
B. Evaluation of FedLay topology

Fig. 7a shows the topology correctness under an extreme situation when 100 new clients join a 400-client FedLay at the same time (10ms in the timeline). The average network latency is set to 350ms. We find the correctness can quickly converge to 1 after 8 seconds in FedLay with degree $d = 6, 8, 10, 12$. Fig. 7b shows the topology correctness under another extreme situation when 100 clients failed from a 400-client FedLay network at the same time (10ms in the timeline). The correctness quickly drops to 64.3%. The remaining clients run NDMP and quickly recover to a correct 300-client FedLay network in 8 seconds. In both situations, NDMP is effective in constructing and maintaining a correct FedLay network. In Fig. 7c, we plot the number of messages sent per client to construct FedLay networks with different sizes. With as many as 500 clients, each client only sends around 30 messages on average.

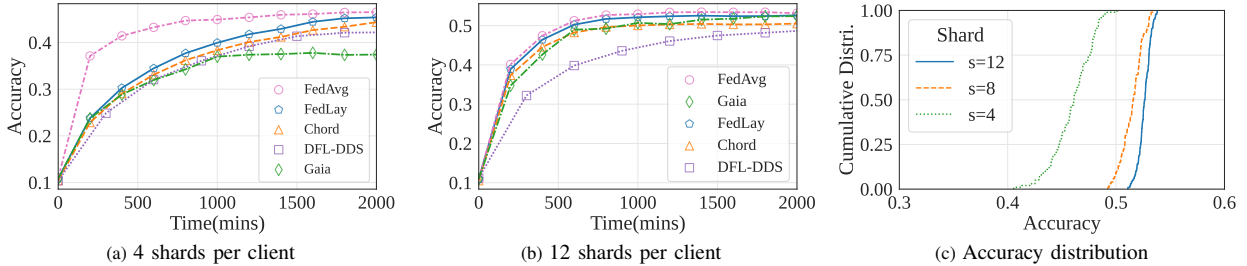
C. DFL model accuracy

Fig. 8 shows the model accuracy of different methods in 6 subfigures, based on real experiments in Amazon EC2. In Figs. 8a-8c we find that FedLay achieves higher accuracy and faster convergence than Gaia and DFL-DDS, even with $d = 4$. Note one communication period for medium-capacity clients is set to 40 minutes in Shakespeare hence there are not many times of model exchanges before convergence. Figs. 8d-8f show the cumulative distribution of the accuracy of all clients at convergence time (150 minutes for MNIST and 1500 minutes for others). We can see that nodes are with similar accuracy levels without any ‘stragglers’.

We evaluate the accuracy of FedLay ($d = 10$), FedAvg, Gaia, and DFL-DDS using medium-scale experiments with 100 clients and show the results in Fig. 9. FedAvg achieves the best accuracy as a centralized FL, which we consider as the upper bound for DFL. The accuracy of FedLay with 10 degrees is only 1.2%, 2.5%, and 0.9% lower than FedAvg on MNIST, CIFAR-10, and Shakespeare, respectively. Other methods have lower accuracy but the differences are not significant. We list



(a) MNIST. Accuracy vs. Time (b) CIFAR-10. Accuracy vs. Time (c) Shakespeare. Accuracy vs. Time
Fig. 9: Model accuracy from 100-client medium-scale experiments. FedAvg is centralized and considered the upper bound.



(a) 4 shards per client (b) 12 shards per client (c) Accuracy distribution
Fig. 10: Accuracy under different non-iid levels for CIFAR-10.

| Task | FedLay | FedAvg | Gaia | Chord | DFLDDS |
|--------|--------|--------|-------|-------|--------|
| MNIST | 90.2% | 92.1% | 89.2% | 88.9% | 87.4% |
| CIFAR | 50.3% | 52.8% | 48.6% | 49.2% | 49.4% |
| Shakes | 45.9% | 46.9% | 44.0% | 44.5% | 44.2% |

TABLE II: Accuracy comparison at convergence. We regard the accuracy of FedAvg as the centralized baseline.

accuracy at convergence time in Table II, along with the default centralized method FedAvg as the baseline.

We evaluate FedLay under different non-iid levels. Each client has a limited number of shards. When each client has fewer shards, the level of non-iid becomes more significant. The default setting is 8 shards per client as shown in previous results such as Fig. 9b, Fig. 10a and Fig. 10b show the accuracy comparison for 4 shards per client and 12 shards per client respectively for CIFAR-10. We find that all DFL methods have slower convergence under more non-iid data (Fig. 10a) but eventually FedLay still achieves similar accuracy as FedAvg, while Gaia and DFL-DDS have lower accuracy. We also show the distribution of accuracy of all clients at the time 2000 in Fig. 10c. When there are 4 shards per client, the distribution is more uneven.

Evaluation of data with biased distribution and locality.

In this set of experiments, 100 clients are divided into 10 groups evenly, and each group possesses 6 out of the total 10 labels in the CIFAR-10 dataset. Each group only has 1 label that is different from the neighboring groups. For example, group 1 has labels 1 to 6; group 2 has labels 2 to 7, etc. and the last group has labels 10, 1, 2, 3, 4, 5. For each client, we sampled 2000 images for each label evenly from the original CIFAR-10 dataset. In Fig.12, we show FedLay has an average of 37.01% improvement over Chord on varying degrees. It also demonstrates FedLay is only 2.0% lower than the theoretical upper bound, a fully connected network. In Fig.13, we show the comparison of the accuracy of FedLay and Chord versus time. Again, FedLay shows much better

convergence compared to Chord.

D. Evaluation of other considerations

Asynchronous communication. We also compare FedLay with synchronous and asynchronous communication in Fig. 11. We find for all three datasets, asynchronous communication can improve both the accuracy and convergence speed, because high-capacity clients do not need to wait for low-capacity ones.

Confidence parameters. Fig. 14 and Fig. 15 shows the accuracy of FedLay with and without confidence parameters for MNIST, compared to simple average. We set $\alpha_d = 0.5$ and $\alpha_c = 0.5$. The results show that FedLay slightly improves the simple average in accuracy.

Accuracy under churn. We show the model accuracy under extreme churn: 50 new clients join a 50-client FedLay network. In Fig. 16, the curves with triangle markers show the accuracy of the initial 50 nodes and the curves with square markers show the accuracy of 50 newly joined nodes. We find that the accuracy of the new nodes quickly converges to a high level due to the high-confidence models from existing nodes. Fig. 17 show that at the join time, the newly joined nodes have very low accuracy and all clients achieve high accuracy eventually.

Computation Cost. In the experiment of 100 nodes training on MNSIT dataset, To reach the accuracy of 88%, The relative computation cost of FedLay is 1.33, compared to 1.53 for Gaia, 2.47 for Chord, and 2.76 for DFL-DDS, with the baseline FedAvg normalized to 1.

E. Scalability

We use large-scale simulations to evaluate the scalability of FedLay, as shown in Fig. 18b. We find that even with up to 1000 clients, FedLay has stable performance in all datasets. In Fig. 18d, we compared the communication cost per client (in MBs) to reach the convergence of FedLay to those of FedAvg, DFL-DDS, and Gaia. Gaia has poor scalability in communication.

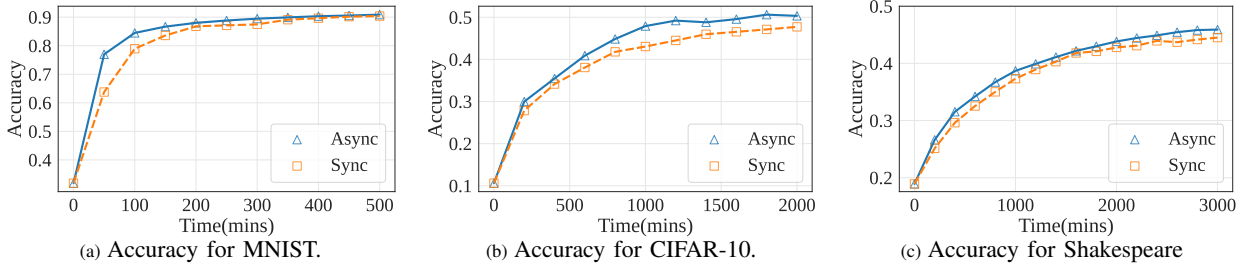


Fig. 11: Model accuracy with synchronous and asynchronous communication.

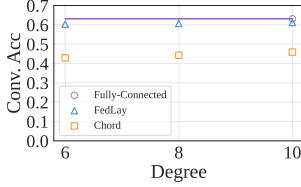


Fig. 12: Converged Accuracy

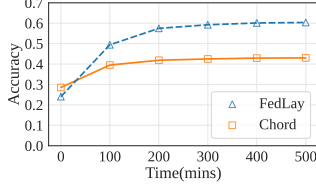


Fig. 13: Accuracy vs Time($d = 8$)

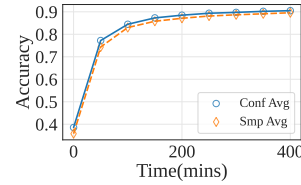


Fig. 14: Accuracy for MNIST, $\alpha_d = 0.5$, $\alpha_c = 0.5$

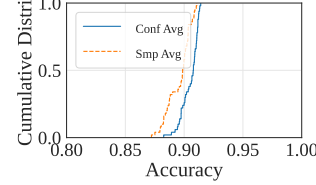


Fig. 15: Accuracy distribution for MNIST

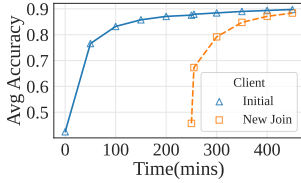


Fig. 16: Accuracy under churn.

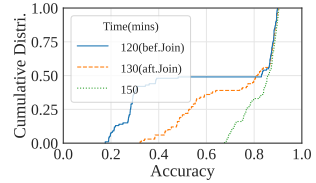


Fig. 17: Accuracy distribution under churn.

V. RELATED WORKS

Decentralized Federated Learning (DFL). Federated learning (FL) [1] is an attractive solution for large-scale ML that allows many clients to train ML models collaboratively without directly sharing training data. However, the central server in FL is a single point of failure and attack. Decentralized Federated Learning (DFL) has been proposed to remove the central server [8], [17], [10]. The overlay network of DFL is a fundamental problem. He *et al.* [8] suggest a few overlay topologies including ring, 2D grid, and complete graph, which either are unable to be constructed in a decentralized way or cause too much communication. GADMM [11] uses a dynamic chain topology and other methods apply clustering-based topologies [13] [12]. Vogels *et al.* [14] analyze model convergence in theory on different topologies including hypercube, torus, binary tree, and ring. Recently Hua *et al.* suggest applying Ramanujan graphs for DFL [16]. Recently, [43], [44] suggest to utilize Blockchain enhance the security and verifiability of DFL.

Other overlay topologies. Overlay topologies have been extensively studied for P2P networks. A well-known category of overlay networks is called distributed hash tables (DHTs),

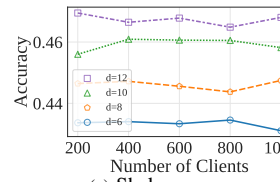
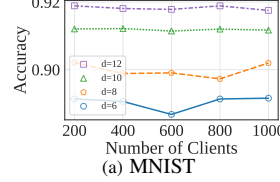
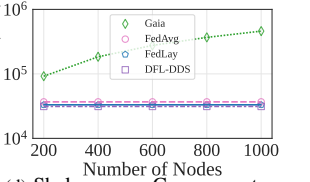
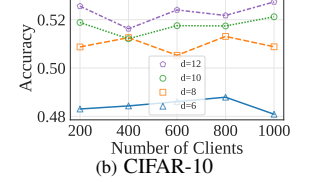


Fig. 18: Results of simulations for large-scale networks



such as Chord [23]. DHTs are proposed to achieve data searching in a decentralized network. Distributed Delaunay triangulation (DT) [25], [26] is designed to achieve greedy routing guarantees and Viceroy [24] is designed for minimizing congestion by overlay routing. Near-random regular topologies have been studied for data center networks with centralized construction [19], [20].

VI. CONCLUSION

This work presents FedLay, the first overlay network for DFL that achieves all the following properties: 1) decentralized construction, 2) fast convergence to accurate models, 3) small node degree, and 4) resilience to churn. We present the detailed designs of the FedLay topology, neighbor discovery and maintenance protocols (NDMP), and model exchange protocol (MEP). We prove that NDMP can guarantee the correctness of a decentralized overlay for node joins and failures. Evaluation results show that FedLay provides the highest model accuracy among existing DFL methods, small communication costs, and strong resilience to churn. In particular, it provides significant model accuracy advantages compared to other decentralized protocols such as Chord, when data are distributed with locality and bias.

ACKNOWLEDGEMENT

We thank our shepherd Yong Liu and the anonymous reviewers for their suggestions and comments. Hua, Pang, Zhang, Yi Liu, and Qian were partially supported by the National Science Foundation (NSF) under Grants CNS-2322919, CNS-2420632, and CNS-2426031. Wang was partially supported by NSF 2436344 and 2219956, and DOE SC0023490 and SC0025589. Yang Liu was partially supported by NSF under Grants IIS-2007951, IIS-2143895 and IIS-2416896.

REFERENCES

- [1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of PMLR AISTATS*, 2017.
- [2] S. P. Karimireddy, S. Kale, M. Mohri, S. Reddi, S. Stich, and A. T. Suresh, "Scaffold: Stochastic controlled averaging for federated learning," in *Proc. of ICML*, 2020.
- [3] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [4] R. Pathak and M. J. Wainwright, "Fedsplit: An algorithmic framework for fast federated optimization," *arXiv preprint arXiv:2005.05238*, 2020.
- [5] S. Reddi, Z. Charles, M. Zaheer, D. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," *arXiv preprint arXiv:2003.00295*, 2020.
- [6] R. Zhou, J. Pang, Z. Wang, J. C. Lui, and Z. Li, "A truthful procurement auction for incentivizing heterogeneous clients in federated learning," in *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2021, pp. 183–193.
- [7] D. A. E. Acar, Y. Zhao, R. Matas, M. Mattina, P. Whatmough, and V. Saligrama, "Federated learning based on dynamic regularization," in *Proc. of ICLR*, 2021. [Online]. Available: <https://openreview.net/forum?id=B7v4QMR6Z9w>
- [8] L. He, A. Bian, and M. Jaggi, "Cola: Decentralized linear learning," *Proc. of NIPS*, 2018.
- [9] T. Wang, Y. Liu, X. Zheng, H.-N. Dai, W. Jia, and M. Xie, "Edge-based communication optimization for distributed federated learning," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 4, pp. 2015–2024, 2021.
- [10] E. T. M. Beltran *et al.*, "Decentralized federated learning: Fundamentals, state-of-the-art, frameworks, trends, and challenges," *IEEE Communications Surveys and Tutorials*, 2022.
- [11] A. Elgabli, J. Park, A. S. Bedi, M. Bennis, and V. Aggarwal, "Communication efficient framework for decentralized machine learning," in *Proc. of IEEE CISS*, 2020.
- [12] A. Bellet, A.-M. Kermarrec, and E. Lavoie, "D-cliques: Compensating for data heterogeneity with topology in decentralized federated learning," in *Proc. of IEEE SRDS*, 2022.
- [13] M. S. Al-Abiad, M. Obeed, M. J. Hossain, and A. Chaaban, "Decentralized aggregation for energy-efficient federated learning via overlapped clustering and d2d communications," *arXiv preprint arXiv:2206.02981*, 2022.
- [14] T. Vogels, H. Hendrikx, and M. Jaggi, "Beyond spectral gap: The role of the topology in decentralized learning," *arXiv preprint arXiv:2206.03093*, 2022.
- [15] Y.-T. Chow, W. Shi, T. Wu, and W. Yin, "Expander graph and communication-efficient decentralized optimization," in *Proc. of IEEE ASIOMAR*, 2016.
- [16] Y. Hua, K. Miller, A. L. Bertozzi, C. Qian, and B. Wang, "Efficient and reliable overlay networks for decentralized federated learning," *SIAM Journal on Applied Mathematics*, vol. 82, no. 4, pp. 1558–1586, 2022.
- [17] T. Sun, D. Li, and B. Wang, "Decentralized federated averaging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
- [18] —, "Stability and generalization of decentralized stochastic gradient descent," in *Proc. of AAAI*, 2021.
- [19] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *Proc. of USENIX NSDI*, 2012.
- [20] Y. Yu and C. Qian, "Space Shuffle: A Scalable, Flexible, and High-Bandwidth Data Center Network," in *Proc. of IEEE ICNP*, 2014.
- [21] S. Boyd, P. Diaconis, and L. Xiao, "Fastest mixing markov chain on a graph," *SIAM review*, vol. 46, no. 4, pp. 667–689, 2004.
- [22] Y. Yu and C. Qian, "Space Shuffle: A Scalable, Flexible, and High-Bandwidth Data Center Network," *IEEE Transactions on Parallel and Distributed Systems*, 2016.
- [23] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proc. of ACM SIGCOMM*, 2001.
- [24] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," in *Proc. of ACM PODC*, 2002.
- [25] D.-Y. Lee and S. S. Lam, "Efficient and accurate protocols for distributed delaunay triangulation under churn," in *Proc. of IEEE ICNP*, 2008.
- [26] S. S. Lam and C. Qian, "Geographic Routing in d -dimensional Spaces with Guaranteed Delivery and Low Stretch," in *Proceedings of ACM SIGMETRICS*, 2011.
- [27] B. M. Waxman, "Routing of multipoint connections," *IEEE journal on selected areas in communications*, vol. 6, no. 9, pp. 1617–1622, 1988.
- [28] J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *Proc. of NIPS*, 2012.
- [29] Y. Hua, J. Pang, X. Zhang, Y. Liu, X. Shi, B. Wang, Y. Liu, and C. Qian, "Towards practical overlay networks for decentralized federated learning," *arXiv preprint arXiv:*, 2024.
- [30] S. Zehabi, S. Hosseinalipour, and C. G. Brinton, "Decentralized event-triggered federated learning with heterogeneous communication thresholds," in *Proc. of IEEE CDC*, 2022.
- [31] J. Cao, Z. Lian, W. Liu, Z. Zhu, and C. Ji, "Hadfl: heterogeneity-aware decentralized federated learning framework," in *Proc. of IEEE DAC*, 2021.
- [32] J. Pang, J. Yu, R. Zhou, and J. C. Lui, "An incentive auction for heterogeneous client selection in federated learning," *IEEE Transactions on Mobile Computing*, 2022.
- [33] S. Kullback and R. Leibler, "On information and sufficiency," *Annals of Mathematical Statistics*, 1951.
- [34] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.
- [35] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-iid data," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 9, pp. 3400–3413, 2019.
- [36] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, vol. 2, 2010.
- [37] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Tech Report*, 2009.
- [38] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," *Tech Report*, 2017.
- [39] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.
- [40] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," 2019.
- [41] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-Distributed machine learning approaching LAN speeds," in *Proc. of USENIX NSDI*, 2017.
- [42] D. Su, Y. Zhou, and L. Cui, "Boost decentralized federated learning in vehicular networks by diversifying data sources," in *Proc. of IEEE ICNP*, 2022.
- [43] X. Zhang, Y. Hua, and C. Qian, "Secure decentralized learning with blockchain," in *2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*. IEEE, 2023, pp. 116–124.
- [44] M. Xu, Z. Zou, Y. Cheng, Q. Hu, D. Yu, and X. Cheng, "Spdl: A blockchain-enabled secure and privacy-preserving decentralized learning system," *IEEE Transactions on Computers*, vol. 72, no. 2, pp. 548–558, 2022.