Towards Scalable Resilient Federated Learning: A Fully Decentralised Approach

Divi De Lacour, Marc Lacoste, Mario Südholt, Jacques Traoré

Towards Scalable Resilient Federated Learning: A Fully Decentralised Approach

Divi De Lacour*†, Marc Lacoste*, Mario Südholt†, Jacques Traoré*

*Orange Innovation
†IMT Atlantique, Inria, LS2N
divi1.delacour@orange.com, marc.lacoste@orange.com
mario.sudholt@imt-atlantique.fr, jacques.traore@orange.com

***Abstract*—Federated Learning (FL) collaboratively trains machine learning models on the data of local devices without having to move the data itself: a central server aggregates models, with privacy and performance benefits but also scalability and resilience challenges. In this paper we present FDFL, a new fully decentralized FL model and architecture that improves standard FL scalability and resilience with no loss of convergence speed. FDFL provides an aggregator-based model that enables scalability benefits and features an election process to tolerate node failures. Simulation results show that FDFL scales well with network size in terms of computing, memory, and communication compared to related FL approaches such as standard FL, FL with aggregators, or FL with election, with also good resilience to node failures.**

***Index Terms*—Federated learning, decentralized learning, pervasive machine learning, edge AI, scalability, resilience**

I. Introduction

Recent years have witnessed the emergence of a cloud-edge continuum providing efficient connectivity to the Cloud for massive numbers of edge devices. This new paradigm also ambitions the pervasive training of machine learning (ML) models for a wide spectrum of data processing applications. While traditional ML uses fully centralized data sets, it requires important communication resources and is subject to security and privacy concerns.

Federated Learning (FL) [1] is a first step to overcome these challenges. FL features a central server that sends the model to clients holding the training data. Clients independently train the model on their local data and send back model updates to the server for aggregation to produce a new model. Unlike centralized learning, FL-related approaches are based on *decentralized architectures* – some of them being *fully decentralized* in the absence of a central node [2].

FL has demonstrated its value for the IoT [3]. But many limitations remain regarding performance, security, privacy, and fairness. Kairouz *et al.* provide a comprehensive review of the corresponding challenges and solutions [4]. In this paper, we focus on FL *scalability* and *resilience* challenges. Scalability means that the server should adapt memory, bandwidth, and computation resources to support an increasing number of client nodes (scaling up). However, a central node limits the number of supported clients, scaling out not being an option due to the network communication overhead. Resilience aims to avoid interruption of training upon failure of the central server or of a significant number of clients. FL architectures may support the failure of client devices, but the central server remains a *single point of failure* [4], [5]. All those elements call for more *decentralized approaches* beyond FL.

Many designs have been explored at the level of models [1], [5]–[7] and frameworks [8], [9]. These approaches investigate different trade-offs between properties such as scalability, resilience, and convergence speed. A key factor is data locality, i.e., the amount of data dependency between neighboring nodes that strongly impacts communication overhead and model accuracy. Unlike strongly-coupled models, fully decentralized or local aggregator-based ones tend to show better resilience and scalability but with slower convergence. Models with less locality may have better performance but are harder to scale or are less likely to tolerate failures.

In this paper, we propose FDFL, a new fully decentralized FL model and architecture that improves standard FL *scalability* and *resilience* based on a peer-to-peer network with no loss of convergence speed. We have chosen to extend the design of Bonawitz *et al.* [7] which adds aggregators to standard FL as proxies between clients and the server for scalability. FDFL features an election process for aggregators and the central server to better tolerate node failures. We also propose a new distributed network architecture supporting the model.

Evaluations of our architecture using the `OMNeT++` simulation framework on a $\sim$ 400k parameter convolutional neural network using CIFAR-10, FMNIST and MNIST datasets show that FDFL scales well compared to other models (e.g., standard FL, FL with aggregators) with constant computing and memory overheads, while preserving convergence speed. FDFL achieves similar resilience to crash failures than standard FL with reasonable failure rates (up to 20 $\%$ of nodes).

The paper is structured as follows. Section II introduces distributed learning. Section III reviews related work. Sections IV and V present the FDFL model and supporting architecture. Sections VI and VII report on experiments and discuss results. Finally, Section VIII concludes and sketches directions for future work.

II. Distributed Learning

ML is traditionally performed on a single computer hosting data and computing power. Some models (e.g., deep learning) require more resources (e.g., memory) beyond those limits. Training may then be performed concurrently.

Distributed learning [10] relies on multiple computing elements (e.g., physical or virtual machines, CPU cores) running concurrently to accelerate training. Beyond consistency and failures challenges, a main hurdle is the communication speed between the entities that are part of the training process.

Distributed ML assumes that all computing resource providers taking part in the training may access the full training dataset. A simple security model is to consider all such providers as trusted by the data providers. In practice, the security model is often more complex. It may include untrusted or mutually distrustful nodes. FL [1] enables distributed learning without requiring data providers to disclose their data: data providers behave as trainers on their own datasets, sharing model updates with a central server.

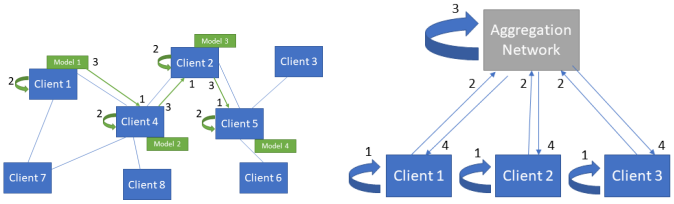
Some main challenges for distributed learning include:

- **Scalability:** performance should remain high when increasing the number of clients rather than node resources.
- **Resilience to failures:** the system should tolerate single or coordinated failures of a reasonable fraction of clients without impact on training.
- **Neighbor dependency**: the topological proximity of network nodes may have various kinds of influence on the evolution of the model during training.
- **Convergence speed:** communication rounds (or steps) to reach a given accuracy should be minimized.
- **Training speed:** latency to reach high accuracy should remain low [11].

Neighbor dependency is closely related to the distribution level and to data locality. Aggregation strategies range from centralized (FL) to decentralized, e.g., tree-based [8], ring-based using parameter servers – partitioning the model over servers holding subsets of the dataset – or fully distributed [10].

Moving from centralized learning to a decentralized setting much widens the design space for models and architectures. Some key dimensions include:

- **Architecture:** communication topology (e.g., star-shaped, peer-to-peer); adaptability (e.g., devices joining/leaving); scalability.
- **Performance:** system resources (computing, memory, bandwidth); convergence speed.
- **Model quality:** high accuracy; low training time.
- **Resilience:** tolerance to failures: single vs. multiple; accidental vs. malicious.
- **Security:** resistance to hostile entities (e.g., devices, routers); confidentiality, integrity, availability threats [12].
- **Privacy:** sensitive/private training data should be protected; inference should not disclose further information.



(a) Gossip models (b) Global gradient models

Fig. 1: Gradient-based decentralised learning

III. Related Work

Gradient-based decentralized learning architectures include:

- *Gossip-based architectures* (see Figure 1a): models are propagated in the network using a *gossip protocol* [6], [13]. A node updates its local model **(1)**, aggregates the models received from neighbors with its own model **(2)**, and sends the result to some of its neighbors **(3)**.
- *Aggregation-based architectures* (see Figure 1b): for each training round, nodes train their model **(1)**, send it to an *aggregator* **(2)** that merges the received models **(3)**, and sends the new model to clients **(4)**.
  - *Local gradient architectures* [5], [14]: each client considers all its neighbors to be aggregators to which it broadcasts its updates. The aggregation result becomes the new client model.
  - *Global gradient architectures* [1], [15]: all clients send their updates to an aggregator which evaluates a global new model.

Table I shows that no architecture gives fully satisfying results. Models with high resilience and scalability tend to show slow convergence, while models with high convergence speeds have moderate scalability and resilience.

Up to our knowledge, there is no model providing at the same time the fast convergence of a global gradient model and the resilience and scalability of fully decentralized approaches. A new learning model is therefore required.

TABLE I: Overview of decentralised learning models

Model	Scalability[1]	Convergence speed[2]	Resilience[1]	Neighbor dependency[1]	Training speed[1]
Global gradient models					
FL [1]	Moderate	High	Moderate	No	High
Scalable FL [7]	High	High	Moderate	No	Moderate
IPLS [15]	Moderate	High	Moderate	No	Moderate
Local gradient models					
CFA [5]	High	Low	High	Yes	High
BrainTorrent [14]	Low	Low	High	Yes	High
Gossip models					
Gossip models [6]	High	Low	High	Yes	Low
Moshpit sgd [16]	High	Moderate	High	No	Moderate

[1]Theoretical analysis [2]Benchmarking

Fault-tolerance vs. Scalability. During training, participants may encounter errors causing loss of network connectivity and node drop out (*crash failures*). Non-malicious failures may happen randomly [4], e.g., a device running out of battery. Failures may also be malicious (*byzantine failures*): a coalition of nodes may tamper with training [17] with many applicable counter-measures [18].

The resilience vs. scalability trade-off has been actively investigated by frameworks. For instance, distributed computing platforms have explored the benefits of extending to ML their architectures for distributed computations [19] or data. With cloud ML hyperscaler solutions, distributed learning is also becoming mainstream. Native distributed ML frameworks [8] generally do not support fault tolerance or have limited scalability due to node availability – some staleness [9] may enhance resilience without affecting convergence. A few frameworks offer sophisticated resilience strategies against stragglers [20] or failing nodes [21], but remain expensive in terms of computational complexity.

Enhancing Security and Privacy. At the hardware level, *Trusted Execution Environments (TEE)* [22] guarantee strong computation and data confidentiality and integrity based on processor-based memory encryption.

At the software level, cryptographic techniques such as *Secure Multi-Party Computation (SMPC)* [23] and *Homomorphic Encryption (HE)* [24] enable respectively multiple agents to compute the result of a common function without sharing local inputs and to compute on encrypted data. Perturbative techniques such as *Differential Privacy (DP)* [25] may also improve privacy by adding noise on nodes during training. Finally, the immutability of *distributed ledgers* [26] is helpful to guarantee distributed integrity in a history of dataset updates.

Such components can be used throughout the architecture to provide different guarantees. The TEE can help to protect client data [17] or to prevent access to the model for clients. Those techniques usually slow down training and may alter the performance of the trained model.

Pervasive FL training will often be deployed on resource-limited devices. The training *execution environment (EE)* can take different forms including virtual machines, containers, or lightweight containers [27], with many tools available [28]. The EE should take into account the diversity of hardware and software, notably hardware accelerators (e.g., GPUs, TPUs) or security components (e.g., TEEs).

IV. Fully Decentralized Federated Learning

We propose the Fully Decentralized Federated Learning (FDFL) model to improve FL scalability and resilience. The model is suitable for P2P networks of edge devices connected to cloud infrastructures.

Model Overview. We adopt the following design principles:

- *Standard FL-inspired* model to preserve the training speed benefits of FL.
- *Aggregator-based model* to increase scalability in terms of computation and communication.
- *No central node in the network* to improve resilience by removing single points of failures in the architecture.
- *Global gradient approach* to enable the best convergence speed without any neighbor dependency.

FDFL applies FL to fully decentralized P2P networks, where nodes know only their neighbors. We apply aggregators to standard FL with no central node to make the model scalable and resilient while preserving training speed. We adopt a global gradient approach for its benefits of convergence speed, model unicity, and absence of neighbor dependency.

The model includes two phases:

- *Role assignment:* three different roles may be assigned to a network node, *client*, *aggregator*, and *server*.
- *Training:* the assigned roles are used to train the model.

Election Process. Regularly (e.g., at constant time intervals), the network elects *aggregators* and a *server* among network nodes. An election should not happen too often to limit communications. But it should allow the network to adapt quickly to failing nodes (e.g., especially the central server).

Training Process. The training process (see Algorithm 1 and Figure 2) adapts and extends that of standard FL [1] similarly to Scalable FL [7] by adding aggregators that behave as proxies between clients and the central server: **(1)** the server dispatches the model to train to clients through the aggregators. **(2)** Clients train the model on their local data. **(3)** Clients send their gradient update propositions to their aggregators. **(4)** The aggregators merge the received updates. **(5)** The results are sent upwards to the server. **(6)** The server merges the updates received from its aggregators. **(7)** The server sends back the new model to clients through the aggregators. Step **7** allows all clients to know the current model in case of a new election.

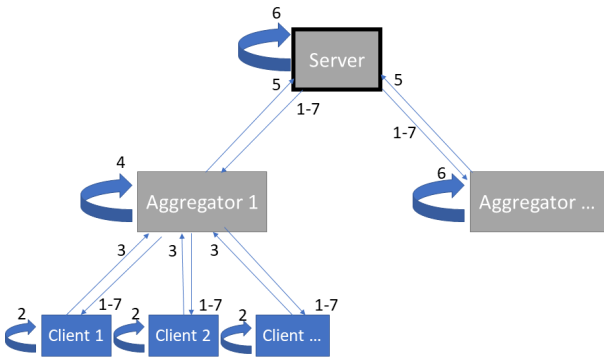


Fig. 2: FDFL training process

In FDFL, using a central server (changing over time according to the election process), instead of a consensus between top-level aggregators (parameter server approach [10]) has benefits in terms of simplicity of implementation and network communications, but decreases resilience. With this design, security and privacy components (e.g., DP, SMPC) may be easily added.

It has been shown that sharing only gradient updates is not enough to guarantee full privacy [12]. In standard FL [1], privacy management is centralized in the server, and some trust should be granted to the server. However, with FDFL, constantly changing aggregators and servers makes the situation different. No node has access to as much data as the central server of standard FL, but more nodes get access to some data on a given node. This decreases the attack surface on a single node, but may increase the number of vulnerable nodes.

Algorithm 1: Training process (adapted from [1]).

n_c: # samples in client c, n_k: # samples per aggregator k, n: dataset size, S_{agg} : set of aggregators, S^k_{client} : sets of clients, μ : learning rate, w: weights

Server:
initialize w_0
for each round t **do**
  for each aggregator $k \in S_{agg}$ *in* || **do**
    $(w^k_{t+1}, n_k) \leftarrow$ AggregatorUpdate(k, w_t)
  $w_{t+1} \leftarrow \sum_{k \in S_{agg}} \frac{n_k}{n} \cdot w^k_{t+1}$
  // Send back new model
  // to clients through
  // the aggregators

AggregatorUpdate(k, w)**:**
// Run on aggregator k
for each client $c \in S^k_{client}$ *in* || **do**
  $(w^c_{t+1}, n_c) \leftarrow$ ClientUpdate(c, w_t)
$w_{t+1} \leftarrow \sum_{c \in S^k_{client}} n_c \cdot w^c_{t+1}$
$n_k \leftarrow \sum_{c \in S^k_{client}} n_c$
return (w_{t+1}, n_k) to server

ClientUpdate(c, w)**:**
// Run on client c
for each local epoch **do**
  for each batch b **do**
    $w \leftarrow w - \mu \nabla \ell(w; b)$
return (w, n_c) to aggregator

To address those challenges, cryptographic (e.g., SMPC, HE), hardware-based (e.g., TEE), or perturbative (e.g., client-level or server-level DP) approaches can be used [12].

V. Architecture

We now describe the FDFL architecture. We present the components that constitute a network node, the supporting election protocol, and the network and failure models.

Node Architecture. We consider the components shown in Figure 3. *Routing* supervises the other node components, and routes messages. *Aggregation* stores and aggregates the proposed models from other nodes. *Training* trains the ML model. *Election* participates in the election of the aggregators and of the server.

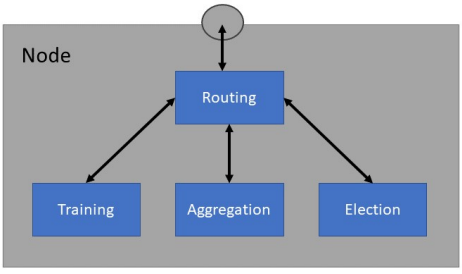


Fig. 3: Node architecture

Such components can be isolated from one another and deployed on different machines. For instance, the training component could be isolated using a TEE to protect the model from curious clients and from gradient poisoning threats.

Election Protocol. *Node election* in P2P networks has been extensively explored, with many protocols providing guarantees of latency, scalability and security [29]. In FDFL, we consider a simple protocol inspired by the Proof of Luck (PoL) scheme [30] – without the TEE which may be added in a second step [31] – to elect a single server and a percentage of aggregators in the network:

1. *Aggregator election:* each client uses a random number generator to elect itself with a tunable α probability. α is set to 2%. The elected aggregators broadcast their PoL score to network nodes.
2. *Server election:* after a pre-defined time, the aggregator with the best PoL score is elected as the server.

Network and Failure Model. We consider a P2P network where nodes only know their direct neighbors. Non-neighboring nodes may communicate by static routing through the network. We consider short-term failures (typically 1s) on the node side. Byzantine failures are not considered at this stage. FDFL makes no further assumption regarding communication between nodes, making it applicable to many communication patterns of distributed ML [10].

Security Modules. Several security modules may be added to protect the components of the architecture. They aim to provide the following protections :

- **Confidentiality –** *Data Confidentiality (DC):* Privacy of client local training dataset. *Model Confidentiality (MC):* Confidentiality of architecture/weights of trained model.
- **Protection against Byzantines –** *Aggregation Integrity (AggI):* The aggregation process is not tampered with. *Byzantine Tolerance (BT):* Model protection against tampering with fake updates. *Backdoor Protection (BackP):* Protection against introduction of backdoors in the model (special Byzantine case)
- **Protection against Sybils –** *Sybil tolerance ST:* Protection against multiple identities created by a single entity.

Table IIa provides an overview of the architecture components on which each class of attack can be applied. Table IIb reviews for each attack type some possible countermeasures. Implementing some of those countermeasures will be a next step beyond this work.

VI. Experimental Results

We evaluate FDFL scalability and resilience w.r.t. other models such as standard FL [1], FL with server election, and FL with aggregators [7].

A. Experimental Setup

We validate FDFL using `OMNeT++`, a `C++` framework for network simulations, e.g. for vehicular systems.We simulate P2P networks with 100 and 1000 nodes and 300 and 3000 links on an Intel Pentium E5300 dual-core 2.6 GHz and 2GB RAM running Linux Ubuntu 20.04. Training of deep learning models

TABLE II: Security – (a) Attack types vs. architecture components; (b) Counter-measures vs. attack types

(a)

Security modules / **Attack type**	**Router**	**Election**	**Aggregation**	**Training**
Data Confidentiality			✓	✓
Model Confidentiality			✓	✓
Aggregation Integrity			✓	
Backdoor Protection			✓	
Byzantine Tolerance	✓		✓	✓
Sybil Tolerance	✓	✓		

(b)

Attack / **Counter-measure**	**DC**	**MC**	**AggI**	**BT**	**BackP**	**ST**
Differential Privacy	✓				✓	
TEE	✓	✓	✓	✓		✓
SMPC	✓					
Reputation System				✓		✓
Filtering				✓		
Redundancy				✓		

is done separately on Google Colab with TensorFlow Federated v0.20 for FL. Models are trained on CIFAR-10, NMIST and FMNIST datasets and evaluated on their accuracy. The model is a convolutional neural network with `3 * (Conv2D` $\rightarrow$ `Maxpooling)` $\rightarrow$ `Dense` $\rightarrow$ `Dense` architecture, with 389,642 parameters. The SGD learning rate is set to 0.001.

B. Scalability

We first analyze the theoretical computing resources, memory, and bandwidth usage during training according to network size before reporting on experimental results. We also discuss the impact of network size on training time.

1) Resource usage: Table IIIa summarizes the node resource usage for different models for transparent (e.g., single-hop) network communications. We assume the memory and the computing time for local training on client devices and the memory to store the model to be constants – the neural network structure is considered static, without compression or structure-specific optimisations.

Our model shows constant computing and memory overheads for aggregation when the network scales. Bandwidth usage outside routing and election is also independent from the network size unlike the other models. We see that the total bandwidth usage for a training step is more important in FDFL because of the use of aggregators behaving as proxies.

2) Communication overhead: We evaluate the network overhead for multi-hop communications, taking into account elections. We consider a single aggregation layer and elections occurring regularly (every 30s). Figure 4 shows the communication overhead for different fractions of aggregators.

We see that outliers get fewer messages when the number of aggregators increases: no node is flooded by messages compared to others. This means that aggregators lift some of the load off the server. Moreover, the median number of messages increases with aggregators. This is due to aggregators behaving as proxies. The election process also has impact on network communications. Finally, the number of messages increases again above a threshold. The network overhead due to the election process then exceeds the benefits of the presence of aggregators. Those trends are confirmed as the network size increases from 100 to 1000 nodes.

TABLE III: Scalability – (a) resource usage; (b) impact on training time

(a)

	Std FL [1]	**Bonawitz [7]**	**FDFL**
Aggregation performance			
Computing	$O(N)$	$O(N/n_{agg})$	$O(A)$
Memory	$O(N)$	$O(N/n_{agg})$	$O(A)$
Bandwidth usage			
Max bandwidth	$O(N)$	$O(N/n_{agg})$	$O(A)$
Total bandwidth	$O(N)$	$O(N/n_{agg})$	$O(N + N/A + N/A^2 \ldots)$
Election performance			
# messages	-	-	$O(N/A)$

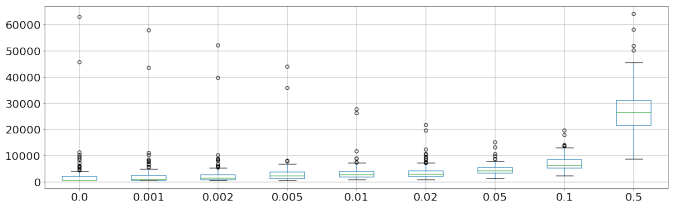
(b)

Model	Training time
Std FL [1]	$2.T_{com} + T_{train} + T_{agg}(N)$
Bonawitz [7]	$4.T_{com} + T_{train} + T_{agg}(N/n_{agg}) + T_{agg}(n_{agg})$
FDFL	$\lceil log_A(N) \rceil . (2.T_{com} + T_{agg}(A)) + T_{train}$

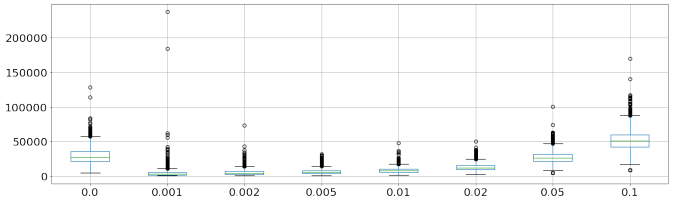
N: # network nodes, A: # clients per aggregator, n_{agg}: # aggregators
T_{train}: training step time on device,
$T_{agg}(A)$: aggregation time for A models,
T_{com}: time for a message to reach its destination

3) Impact of network size on training time: Table IIIb shows the impact of the network size on training time. The training time is more important in FDFL than for other models due to the presence of aggregators – although for some cases such as FL where $T_{agg}(N) = O(N)$, logarithmic speedups may be achieved.

A trade-off may be found between training time, computing and memory overheads of a single node. Reducing the number of aggregation layers improves training time and reduces the communication overhead and the cost of the election process, but puts a higher toll on the aggregators, increasing significantly the upper bound on network traffic.



(a)



(b)

Fig. 4: Scalability: number of exchanged messages vs. fraction of aggregators – (a) 100 nodes; (b) 1000 nodes
bar: median, box: quartiles, whiskers: data range (excl. outliers), dots: outliers

C. Resilience

We evaluate as well the resilience of FDFL to non-byzantine failures: nodes may fail randomly for short, random periods of time. Resilience is assessed for a single layer of aggregators representing 2% of network nodes by training models and computing their accuracy. We simulate short failures of around 1s for the models with and without aggregation and election processes shown in Table IV.

TABLE IV: Considered Models

Model	Aggregators	Election
FL [1]		
Bonawitz [7]	✓	
Election FL		✓
FDFL	✓	✓

1) Influence of aggregators and election: Results for CIFAR-10 are shown in Figure 5 for $f = 0.1$, where f is the average fraction of failing nodes. We observe that FDFL has about the same resilience as standard FL (see Figure 5a). Also, we find that separately, aggregation and election protocols have an impact on resilience, but together, they enhance it (see Figures 5b and 5c).

The presence of aggregators reduces training performance in the presence of failures (see Figure 5b). This is due to aggregators behaving as proxies, which increases the distance to the server and the likelihood of messages being dropped.

With an election process, some nodes may not be aware of the identity of the newly elected server, this information being lost due to failures. This reduces the number of active participants and model accuracy (see Figure 5c).

However, when using aggregators and election together, FDFL improves resilience compared to using them separately. With aggregators, it is easier for a client to know at least one aggregation node and to participate in training. With an election, the regular change of aggregators increases the diversity of participating clients and training samples, improving accuracy. Table Va provides accuracy benchmarks for other datasets, but the conclusions are broadly similar.

TABLE V: Accuracy for different datasets – (a) model comparison (b) impact of failures

(a)

Model / Dataset	FL f=0	FL	FDFL	Bonawitz	Election FL
CIFAR-10	0.6361	0.6172	0.6104	0.6084	0.6071
MNIST	0.9891[1]	0.9894	0.9871	0.9879	0.9863
FMNIST	0.8828[2]	0.8954	0.8858	0.8914	0.8873

(b)

Model / Dataset	FL f=0	FDFL f=0.1	f=0.2	f=0.3	FDFL f=0.1	f=0.2	f=0.3
CIFAR-10	0.6361	0.6172	0.5965	0.5668	0.6104	0.5623	0.4584
MNIST	0.9891[1]	0.9894	0.9882	0.9837	0.9871	0.9858	0.9718
FMNIST	0.8828[2]	0.8954	0.8876	0.869	0.8858	0.8816	0.8346

[1] 131 epochs [2] 104 epochs

2) Influence of failure rate: We see that performance is slightly lower for our model than for FL as the failure rate increases (see Figure 5d and Table Vb) but remains quite acceptable. Beyond $f = 0.2$, performance degrades further, although still good for MNIST and FMNIST, perhaps due to less complex datasets. FDFL seems thus to be well adapted to network environments with reasonable failure rates.

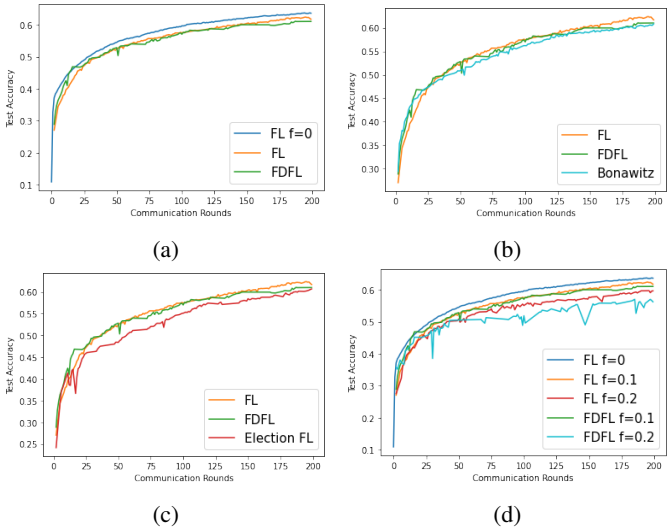


Fig. 5: Resilience (CIFAR-10) – influence of (a) model (b) aggregators (c) election (d) number of failures

3) Scalability vs. Resilience Trade-off: Benchmarks highlighted that elections tend to increase bandwidth usage. A trade-off should be found between having regular elections to improve resilience but within reasonable bounds to preserve bandwidth usage and scalability.

VII. Discussion

We observed that FDFL improves FL scalability, as shown by simulation results for networks with 100 and 1000 nodes. Further experiments are needed for networks with multiple layers of aggregators and when considering an evolution of the number of participants during training.

FDFL shows good resilience against short-term and non malicious failures for simple datasets. Additional research is needed to investigate more realistic failure models, such as long-term and malicious (i.e., Byzantine) failures [32], [33].

Several reactions of the network to a node failure are possible, depending on its role:

- **Server failure:** training of the entire network is stopped until the server recovers or another server is elected in the next election round.
- **Aggregator failure:** data of the aggregator's clients are not taken into account until the aggregator recovers or clients select another aggregator in the next election round.
- **Client failure:** client data are not taken into account until the first training step after recovery. If a new election happens during the failure period, the client has to wait for the next election to participate again.

When they fail, aggregators have more impact than a normal client on the quality of the model: they may drop gradient updates of multiple clients and the network may fork into

two branches training concurrently, merging back only at the next election round. This reduces the number of clients active during training, and the diversity of training data.

Though, the model remains very simple and datasets such as CIFAR-10 not particularly demanding. Training datasets with more complex data would allow to highlight the influence on accuracy associated with a loss in training data diversity [34].

VIII. Conclusion

This paper presented FDFL, a new fully decentralized FL model and architecture combining aggregators and election process that improves the scalability and the resilience of standard FL while preserving convergence speed.

Future work includes enhancing security and privacy [35] by adding new components to the architecture, e.g., SMPC, HE, TEE [36], and DP [37]. For instance, SMPC and TEE could be efficiently and simply integrated, e.g., the SMPC protocol of [31] combines well with the PoL election process used in FDFL assuming TEE-protected nodes.

Another direction is to improve performance, e.g., by exploring the influence of multiple aggregator layers w.r.t. the number of clients or by relaxing model unicity [38].

IX. Acknowlegments

We would like to thank the anonymous reviewers for their many insightful comments. This research was partially supported by the CRYPTECS project which has received funding from the French ANR through grant ANR-20-CYAL-0006 and from the German BMBF through grant 16KIS1441.

References

[1] H. B. McMahan et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data," *arXiv:1602.05629*, 2017.

[2] P. Bellavista et al., "Decentralised Learning in Federated Deployment Environments: A System-Level Survey," *ACM Comput. Surv.*, vol. 54, no. 1, pp. 1–38, 2021.

[3] T. D. Nguyen et al., "DÏoT: A Federated Self-learning Anomaly Detection System for IoT," in *ICDCS'19*.

[4] P. Kairouz et al., "Advances and Open Problems in Federated Learning," *Foundations and Trends in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[5] S. Savazzi, M. Nicoli, and V. Rampa, "Federated Learning with Cooperating Devices: A Consensus Approach for Massive IoT Networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4641–4654, 2020.

[6] I. Hegedűs, G. Danner, and M. Jelasity, "Gossip Learning as a Decentralized Alternative to Federated Learning," in *DAIS'19*.

[7] K. Bonawitz et al., "Towards Federated Learning at Scale: System Design," *arXiv:1902.01046*, 2019.

[8] A. Agarwal et al., "A Reliable Effective Terascale Linear Learning System," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1111–1133, 2014.

[9] E. P. Xing et al., "Petuum: A New Platform for Distributed Machine Learning on Big Data," *IEEE Trans. on Big Data*, vol. 1, no. 2, pp. 49–67, 2015.

[10] J. Verbraeken et al., "A Survey on Distributed Machine Learning," *ACM Computing Surveys*, vol. 53, no. 2, 2020.

[11] H. Zhang et al., "Poseidon: An Efficient Communication Architecture for Distributed Deep Learning on GPU Clusters," in *USENIX ATC'17*.

[12] N. Rodríguez-Barroso et al., "Survey on Federated Learning Threats: Concepts, Taxonomy on Attacks and Defences, Experimental Study and Challenges," *arXiv:2201.08135*, 2022.

[13] A. Agrawal et al., "On Decentralizing Federated Learning," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020.

[14] A. G. Roy et al., "BrainTorrent: A Peer-to-Peer Environment for Decentralized Federated Learning," *arXiv:1905.06731*, 2019.

[15] C. Pappas et al., "IPLS : A Framework for Decentralized Federated Learning," *arXiv:2101.01901*, 2021.

[16] M. Ryabinin et al., "Moshpit sgd: Communication-efficient decentralized training on heterogeneous unreliable devices," in *NIPS'21*.

[17] S. Prakash et al., "Byzantine-Resilient Federated Learning with Heterogeneous Data Distribution," *arXiv:2010.07541*, 2021.

[18] L. Lyu et al., "Privacy and Robustness in Federated Learning: Attacks and Defenses," *arXiv:2012.06337*, 2020.

[19] Chu et al., "Map-Reduce for Machine Learning on Multicore," in *NIPS'06*.

[20] H. Cui et al., "Exploiting Bounded Staleness to Speed up Big Data Analytics," in *USENIX ATC'14*.

[21] M. Li et al., "Scaling Distributed Machine Learning with the Parameter Server," in *ODSI'14*.

[22] F. Mo et al., "PPFL: Privacy-preserving Federated Learning with Trusted Execution Environments," *arXiv:2104.14380*, 2021.

[23] K. Bonawitz et al., "Practical Secure Aggregation for Privacy-Preserving Machine Learning," in *CCS'17*.

[24] Zhou et al., "Privacy-Preserving Federated Learning in Fog Computing," *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 10 782–10 793, 2020.

[25] M. Abadi et al., "Deep Learning with Differential Privacy," in *CCS'16*.

[26] K. Pandl et al., "On the Convergence of Artificial Intelligence and Distributed Ledger Technology: A Scoping Review and Future Research Agenda," *IEEE Access*, vol. 8, pp. 57 075–57 095, 2020.

[27] A. Haas et al., "Bringing the Web up to Speed with WebAssembly," in *PLDI'17*.

[28] TensorFlow Federated, https://www.tensorflow.org/federated/.

[29] Y. Xiao et al., "A Survey of Distributed Consensus Protocols for Blockchain Networks," *IEEE Comm. Surveys & Tutorials*, vol. 22, no. 2, pp. 1432–1465, 2020.

[30] M. Milutinovic et al., "Proof of Luck: an Efficient Blockchain Consensus Protocol," in *1st Workshop on System Software for Trusted Execution (SysTEX)*, 2016.

[31] R. Bahmani et al., "Secure Multiparty Computation from SGX," in *Financial Cryptography and Data Security*, 2017.

[32] F. Elhattab et al., "Robust Federated Learning for Ubiquitous Computing through Mitigation of Edge-Case Backdoor Attacks," *Proc. ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 4, pp. 1–27, 2022.

[33] I. Messadi et al., "SplitBFT: Improving Byzantine Fault Tolerance Safety Using Trusted Compartments," in *ACM/IFIP International Middleware Conference (MIDDLEWARE)*, 2022.

[34] J. O. du Terrail et al., "FLamby: Datasets and Benchmarks for Cross-Silo Federated Learning in Realistic Healthcare Settings," in *NeurIPS'22*.

[35] F. Mo, Z. Tarkhani, and H. Haddadi, "SoK: Machine Learning with Confidential Computing," *arXiv:2208.10134*, 2022.

[36] A. A. Messaoud et al., "Shielding Federated Learning Systems against Inference Attacks with ARM TrustZone," in *ACM/IFIP International Middleware Conference (MIDDLEWARE)*, 2022.

[37] E. Cyffers and A. Bellet, "Privacy Amplification by Decentralization," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2022.

[38] C. Xie, S. Koyejo, and I. Gupta, "Asynchronous Federated Optimization," *arXiv:1903.03934*, 2020.