2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) | 978-1-6654-6367-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/CloudCom55334.2022.00027

FedTeams: Towards Trust-Based and Resource-Aware Federated Learning

Dorde Popovic*§ Hend K. Gedawy† Khaled A. Harras†

*Qatar Computing Research Institute, Hamad Bin Khalifa University
†Computer Science Department, Carnegie Mellon University
*dpopovic@hbku.edu.qa, †hkg@andrew.cmu.edu, †kharras@cs.cmu.edu

***Abstract*—Federated Learning (FL) has enabled Machine Learning (ML) applications to capture a larger spectrum of data by allowing such data to remain on-device, a desirable privacy guarantee in many applications. However, the highly iterative nature of FL optimization algorithms requires low-latency and high-throughput connections to clients. Unfortunately, realistic FL training scenarios include heterogeneous clients that are restricted by computation and communication, thereby slowing down or even failing FL training. In this paper, we propose FedTeams; a trust-based and resource-aware FL system that minimizes training latency, while improving accuracy. To achieve this, we mitigate the risk of straggling and weakly-connected clients by leveraging social trust and allowing these clients to offload their data to more powerful trusted peers that can train on their behalf. In specific, we formulate and solve an optimization problem that leverages the FedTeam's trust graph and client resource information to optimize the distribution of training and minimize training latency. We evaluate FedTeams in a simulated environment, demonstrating up to a 81.6% decrease in training latency and 11.2% increase in global model accuracy when compared to existing state-of-the-art solutions.**

***Index Terms*—Federated Learning, Internet of Things, Edge Computing, Trust, Privacy**

I. Introduction

Data privacy in Machine Learning (ML) applications has proven to be a major concern in various domains like health, finance, and other applications using forms of personal user data [1]–[4]. To address these concerns, Federated Learning (FL) was introduced [5], proposing that a global model can be distributed to all clients containing relevant training data. These clients can then train their model locally and independently on their own data, only ever sharing model parameter updates. These updates are aggregated by an FL server and used to iteratively update the global model. The hope with FL is that it would ultimately enable access to more data than ML due to this promise of privacy, which in turn leads to higher model accuracy and faster convergence.

Minimizing training latency is a fundamental challenge in FL [6]. This challenge stems from the fact that optimization algorithms (e.g., stochastic gradient descent) that FL inherits from ML are highly iterative, require high-throughput, and demand low-latency data access during training [6]. Unlike ML, data in the context of FL resides on client devices which run training locally and regularly communicate model updates to the FL server over a network with significantly higher delays [7]. Additionally, the natural occurrence of resource-constrained or poorly connected clients introduces stragglers, which further slow down or entirely halt FL training [6].

As such, recent work in FL has explored various approaches to reduce training latency (Section II). These approaches include: (a) introducing intermediary aggregation nodes [8]; (b) devising mechanisms for client selection and training scheduling [9]; (c) modifying model complexities that allow heterogeneous clients train models scaled to their capabilities [10]; (d) clustering clients in order to speedup model convergence [11], [12]; (e) optimizing the underlying network factors and resources for FL, thereby mitigating network communication delay and client connectivity issues [13]–[15]. All these approaches, however, are constrained by the fundamental assumption that training must take place at individual participating nodes in FL pools in order to preserve privacy.

We argue for novel solutions that offer comparable privacy by leveraging contexts where trust may exist amongst even a small subset of participating nodes. Trust between devices may exist in situations such as homes, neighborhoods, and classrooms, permitting data to be offloaded between trusted clients. The establishment of these trust relations is ultimately a function of the nature of the application. As such, trust relations can be deduced based on device ownership (i.e., multiple devices owned by one user), owner-to-owner relationships, manual user configuration, or various forms of device-to-device contact and proximity thresholds [16], [17].

In this paper, we propose FedTeams, a middle-ground training solution between FL and centralized ML, that leverages trust relations between clients to minimize FL training latency (Section III). FedTeams is composed of a group of clients (FedPlayers) partaking in FL training that has been initiated by an FL server. A single FedPlayer is elected as the FedCaptain, responsible for maintaining awareness of FedPlayer resources and trust relationships, and using this information to produce the training assignment. The training assignment includes assigning a subset of FedPlayers (FedStarters) the responsibility of aggregating data and training their local model on this data. The remaining FedPlayers (FedSubs) do not train locally and are assigned a FedStarter that they trust, to which they offload their data for training.

The core functionality of the FedTeams system is the creation of the training assignment (Section IV). This task is formulated as an optimization problem that strives to minimize training latency under multiple constraints related to client

§The author did this work while he was at Carnegie Mellon University.

device energy, storage, and trust relations. We leverage trust to enable FedTeams to select highly-connected, resource-rich, and task-dedicated clients to lead training. FedTeams simultaneously excludes high-risk straggler clients whilst still including their data in FL training. Alongside minimizing training latency, the inclusion of data from potential stragglers and previously ineligible clients supports the core FL goal of increasing data coverage to speed up the convergence of the global model [5]. We stress that FedTeams also offers a set of secondary benefits that include reduced total energy expenditure across all clients and increased data centralization that leads to higher model accuracy. The latter is especially useful in scenarios where data across the clients is not independent and identically distributed (Non-IID). There is a greater gain to be achieved in aggregating and training on data whose distribution and resulting optimization direction becomes closer to that of the global data set and model.

We evaluate FedTeams, in terms of training latency, accuracy, and energy consumption, using simulation (Section V). We assess the system performance while varying data distribution, FedTeams number/sizes, degrees of trust, and levels of client resource heterogeneity. Our results demonstrate latency savings of up to 81.6% compared to the Standard [5] and Hierarchical [18] FL baselines. Our system also achieves up to a 11.2% accuracy improvement and 90.1-95.0% reduction in total energy consumption across all clients.

II. Related Work

Past research has explored minimizing training latency by mitigating the negative impact of the heterogeneity of client resources. FogFL [8] proposes placing fog nodes in geospatial locations that ensure client end-device coverage, thereby reducing communication latency and energy consumption of resource-constrained clients. FedCS [9] takes into consideration the resource capability and availability of each client, selecting clients and scheduling training deadlines that minimize latency and avoid congestion. HeteroFL [10] proposes modifying the global model by varying the width of its hidden channels. This produces local models of varying complexity that utilize a significantly smaller number of local model parameters. Models are distributed to clients with heterogeneous resources, such that each client receives a model of complexity scaled to its capabilities. This reduces the likelihood of straggler clients and enable clients with heterogeneous resources to meet the same reporting deadline. We note that the approach presented in HeteroFL [10] can be integrated within our solution, aiding in lowering the barrier of entry to FL by allowing a greater spectrum of clients to partake in training when trust-based offloading is not feasible.

Various techniques of latency minimization through client clustering have been studied. Semi-FL [11] proposes building clusters that leverage efficient Device-to-Device communication technologies for iterative training of a cluster model, employing multiple clusters in training a global model. FedGroup [12] builds clusters based on similarity between client optimization directions, thereby speeding up the convergence of the global model and thus reducing overall training latency.

Some researchers have tackled latency minimization through optimizing wireless network factors and resources for FL training [13]–[15]. Yang et al. [13] explore the trade-off between local computation delay and wireless communication delay, and propose a bisection-based algorithm for obtaining the optimal solution. Meanwhile, Samarakoon et al. [14] focus on wireless network policies that ensure low-latency FL in the specific context of vehicular communication networks.

Our work aims to minimize training latency in heterogeneous FL training settings by challenging the binary assumption that either all training takes place on client device (FL), or all training takes place on a centralized server (Centralized ML). We observe common training scenarios (i.e., home, classroom, office, etc.,) where trust exists between clients, and we propose leveraging this trust to allow data offloading between trusted clients. This flexibility makes it possible to optimize the selection of training clients that minimize training latency whilst still including all available training data to maintain/improve model accuracy.

III. System Architecture

Figure 1 presents a high-level view of our system architecture, which consists of two main components: an FL Server and a set of FedTeams. The FL Server manages training the global model across the FedTeams, which, in turn, manage their local model training and aggregation.

The **FL Server** has the *Training Manager*, which is responsible for managing global model training across FedTeams. It starts the training by retrieving the global model configuration and sample data from persistent storage to create an initial global model. It then admits FedTeams registering for training and initiates the global FL training rounds. Each global round begins with the FL server distributing global model parameters to registered FedTeams. Each of these FedTeams runs multiple local FL training rounds and reports to the FL Server the resulting model parameter updates. The global round ends once all FedTeams have reported their updates. FL training ends once the global model accuracy converges or a preset maximum number of rounds is reached.

A **FedTeam** is a collection of clients capable of communicating with each other and willing to participate in FL training. We note that our system architecture abstracts the FedTeams component, such that from the perspective of the FL Server, communication with the FedTeam is identical to communication with a standard FL Client. Each FedTeam is comprised of (1) a FedCaptain and (2) a set of FedPlayers.

The **FedCaptain** is responsible for maintaining awareness of client resources and trust relationships, and using this information to solve the training assignment problem. One of the clients in a FedTeam is selected to be the FedCaptain through a classical distributed election algorithm [19], while all other clients become FedPlayers. The FedCaptain could be a server, desktop, laptop, or an IoT/mobile client, especially given the resources available in modern IoT/mobile devices [20], [21].

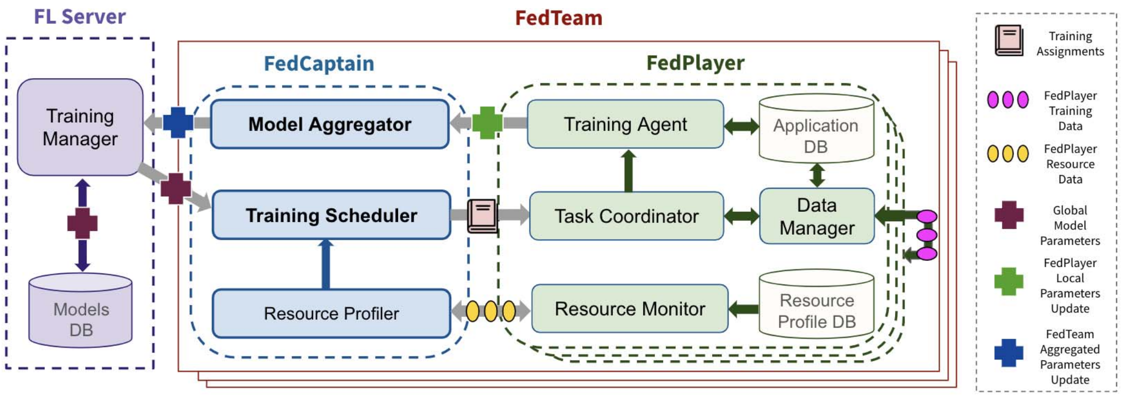


Fig. 1. System Architecture

Once elected, the *Resource Profiler* receives information on the communication throughput, computational capacity, energy availability, storage availability, and list of trusted clients from each FedPlayer in the FedTeam. We note that this collection of resource information is independent from client data and does not affect client privacy in the context of FL. The *Training Scheduler* receives the FedPlayer profiles and global model configuration, and uses this information to build the training assignment graph. This graph defines which FedPlayers will be executing local training (FedStarters), which FedPlayers will be offloading their data (FedSubs), and which FedStarters these FedSubs will be offloading their data to. The Training Scheduler informs each FedPlayer of its role. Each FedStarter is informed about FedSubs from which it needs to aggregate data while each FedSub is informed of its assigned FedStarter. Once all FedStarters complete local training, the *Model Aggregator* aggregates their model parameter updates and sends the result to the FL Server.

The **FedPlayer** holds training data and either trains on this data or offloads it to another FedPlayer to train on. FedPlayers contain a *Resource Monitor* that is responsible for monitoring and sending the aforementioned FedPlayer resource and trust information to the FedCaptain. The *Task Coordinator* retrieves the training assignment graph from the FedCaptain. If assigned the role of **FedStarter**, the *Task Coordinator* instructs the *Data Manager* to aggregate training data from all FedSubs that have been assigned to the FedStarter. This data is compiled alongside the FedStarter's local training data and the global model configuration, which are altogether provided to the *Training Agent*. This agent performs the local training on the model and sends the resulting model parameter updates to the FedCaptain. Otherwise, if assigned the role of **FedSub**, the *Task Coordinator* simply instructs the *Data Manager* to offload local training data to the assigned FedStarter.

IV. Training Assignment Optimization

In this section, we formulate the problem of creating a training assignment as an optimization problem, operating under client trust and resource constraints, with the objective of minimizing training latency. We start by presenting our system model, followed by formalizing the problem definition and developing a heuristic algorithmic solution.

TABLE I
FedTeams symbols

Symbol	Description
C_i	Computational capacity of FedPlayer i
T_i	Comm. throughput between FedCaptain and FedPlayer i
T_{ij}	Comm. throughput between FedPlayers i and j
$\mathcal{E}_i$	Energy available for training at FedPlayer i
$\mathcal{E}_{\text{comp},i}$	Computation energy consumption rate of FedPlayer i
$\mathcal{E}_{\text{comm},i}$	Comm. energy consumption rate of FedPlayer i
S_i	Storage available at FedPlayer i
D_i	Size of training data set at FedPlayer i
$L_{i,j}$	Processing load introduced to FedPlayer j by training model on data from FedPlayer i
G_i	Set of FedPlayers trusted by FedPlayer i
F_{start}	Set of FedStarters
F_{sub}	Set of FedSubs
o_{ij}	True (1) if FedPlayer i offloads data to FedPlayer j
n	Number of global FL rounds
t	Number of FedTeams
p	Number of FedPlayers per FedTeam
w	Size of model parameter updates

A. System Model

Table I summarizes the parameters relevant to our system model. We assume that all FedPlayers are willing to participate in the global FL training process. The FL Server communicates with the t FedCaptains using a cellular channel interface, while each individual FedCaptain and team of p FedPlayers leverage a close proximity communication channel to share resource, assignment, training, and model parameter data.

We assume that trust between FedPlayers is binary and provided as input to the system. Additionally, we assume that the data of all FedPlayers is correctly labeled, however noting that techniques have been proposed for overcoming mislabeling and other data noise in FL [22]. We make no assumptions about the data being independently and identically distributed (IID), instead considering both IID and non-IID settings.

Finally, we make no assumptions on the stability of FedPlayer resource availability and task dedication. Thus, we design our system to be resilient in settings with unpredictable FedPlayers. Specifically, the FedCaptain dynamically tracks the resource profiles of FedPlayers and the current location of all training data within the FedTeam. This allows it to not only detect stragglers when they arise, but also reallocate data from these stragglers to high-performing FedPlayers.

B. Problem Formalization

The main objective of a FedCaptain's *Training Scheduler* is to create an optimized assignment that minimizes the team's training time at each round. Since we aim to include parameter updates from all FedPlayers in the current FedTeam, the team's round completion time is equivalent to the time taken by the slowest FedPlayer. This yields the objective function:

$$\text{Minimize } \max \mathcal{T}_i \tag{1}$$

where $\mathcal{T}_i$ is the completion time of a single FedTeam round of training by the i^{th} FedPlayer.

For each FedSub i, this completion time is simply the time required to offload data to its assigned FedStarter:

$$\mathcal{T}_i = \underbrace{\sum_{j \in F_{start}} o_{ij} \cdot \frac{D_i}{T_{ij}}}_{\text{Data Offloading}} \tag{2}$$

where the data offloading time consists of the time spent retrieving data from the FedSub's *Application DB* and sending it to the FedStarter.

For each FedStarter j, this completion time is the time required to aggregate data from all of its assigned FedSubs, complete local training of the model, and communicate model parameter updates to the FL Server:

$$\mathcal{T}_j = \underbrace{\sum_{i \in F_{sub}} o_{ij} \cdot \frac{D_i}{T_{ij}}}_{\text{Data Aggregation}} + \underbrace{\frac{L_{j,j} + \sum\limits_{i \in F_{sub}} o_{ij} \cdot L_{i,j}}{C_j} + \frac{w}{T_j}}_{\text{Model Training}} \tag{3}$$

where the data aggregation time consists of the time spent by the FedStarter waiting and receiving data from all assigned FedSubs. The model training time consists of time spent preparing the data, loading the model, training for a set number of epochs, and sending parameter updates to the FL server.

Alongside our objective function, we also operated under a set of constraints that must be satisfied by our assignment.

Model Bias Prevention: A FedSub i may only offload its data to a single FedStarter, as offloading to multiple FedStarters can result in model bias towards repeated data; i.e. $\sum\limits_{j \in F_{start}} o_{ij} \leq 1$.

FedPlayer Energy Limitation: The tasks of offloading/aggregating data, running training, and communicating model parameter updates should not cause any FedSub i or FedStarter j to exceed its energy threshold. That is: $\sum\limits_{j \in F_{start}} o_{ij} \cdot D_i \cdot \mathcal{E}_{comm,i} \leq \mathcal{E}_i$.

$\sum\limits_{i \in F_{sub}} o_{ij} \cdot (D_i \cdot \mathcal{E}_{comm,j} + L_{i,j} \cdot \mathcal{E}_{comp,j}) + w \cdot \mathcal{E}_{comm,j} \leq \mathcal{E}_j$.

FedStarter Storage Limitation: The data aggregation should not cause any FedStarter j to exceed its available storage; i.e. $\sum\limits_{i \in F_{sub}} o_{ij} \cdot D_i \leq S_j$.

Algorithm 1 FedTeam Training Assignment Algorithm

```
 1: variables definition
 2:     $GW$, $LW$: the global and local model weights, respectively
 3:     $TR$: assigned internal training rounds in a FedTeam
 4:     $FT$: a FedTeam
 5:     $FT.\{P\}$, $FT.\{FS\}$: the set of FedPlayers and FedStarters in FedTeam (FT)
 6:     $P.\{TP\}$: the set of all FedPlayers that FedPlayer (P) trusts
 7:     $P.d$: the data size at FedPlayer (P)
 8:     $FT.t$: max time bin capacity across FedPlayers in FedTeam (FT)
 9: end variables definition
10: procedure TRAIN(FT, TR)
11:     for i in TR do
12:         $GW \leftarrow receiveGlobalWeights()$
13:         $FT.\{FS\} \leftarrow FT.createTrainingAssignments()$
14:         $sendGlobalWeights(GW, FT.\{FS\})$
15:         $\{LW\} \leftarrow waitForFedStartersLocalWeightUpdates()$
16:         $FT.LW \leftarrow aggregateLocalWeightUpdates(\{LW\})$
17:         $sendAggregatedWeightsToFLServer(FT.LW)$
18:     end for
19: end procedure
20: procedure CREATETRAININGASSIGNMENTS()
21:     $\{FS\} \leftarrow \{\}$                                   ▷ FedStarters
22:     $FT.t \leftarrow 0$                          ▷ Initialize Time Bins Capacity
23:     $\{SP\} \leftarrow sortFedPlayersDescending(FT.\{P\}, key = P.d)$
24:     for $P$ in $\{SP\}$ do
25:         $\{EP\} \leftarrow filterEligiblePlayers(P.\{TP\})$
26:         $EPT = \{\}$
27:         for $EP$ in $\{EP\}$ do
28:             $tt \leftarrow estimateTrainingTime(EP, P)$
29:             $EPT.append(EP, tt)$
30:         end for
31:         $EP', tt' \leftarrow EPT.findPlayerWithMinTimeBin()$
32:         $EP'.assignTraining(P)$
33:         $P.assignDataOffloading(EP')$
34:         $\{FS\}.append(EP')$
35:         $EP'.t \leftarrow tt'$
36:         $EP'.updateEnergyAndStorageBins()$
37:         if $tt' > FT.t$ then
38:             $FT.t \leftarrow tt'$
39:         end if
40:     end for
41: end procedure
```

FedSub Trust Requirement: A FedSub i may only be assigned to offload its data to a FedStarter j if the FedSub client trusts the FedStarter client; i.e. $o_{ij} \leftrightarrow j \in G_i$.

C. FedTeam Training Assignment Algorithm

The problem that we have formalized is an instance of the more general mixed integer programming problem. This is a known NP-complete problem, with a known polynomial-time reduction to the bin-packing problem [23].

Hence, we adapt a known approximation algorithm for bin-packing to our problem. We consider the training time required for data chunks belonging to FedPlayers to be the packing items, and the FedPlayers to be time bins. The training time of the data chunks (items size) is tied to the client device's resources. By applying the first-fit-decreasing heuristic approach to this offline version of the problem, we attain an assignment of items to bins, and therefore the assignment of FedPlayer data chunks (FedSubs) to FedPlayers (FedStarters). Another key change in our algorithm is the measure of a bin's capacity, which we select to be the current value of our objective function (i.e., completion time of current slowest FedPlayer). The result of this is a load-balancing variant of the problem where we minimize the completion time of the slowest FedPlayer.

Algorithm 1 presents the detailed pseudocode executed by the *Training Scheduler* component of the FedCaptain. The

Train module (lines 10-19) is executed by the FedCaptain at each FL round. The inputs to this module are the registered FedTeam and the number of training rounds. It executes the following steps: computing the training assignments (line 13), retrieving global weights from the FL server and forwarding them to all FedStarters (lines 12 & 14), waiting for all FedStarters to aggregate data from their assigned FedSubs and complete their local training rounds (line 15), aggregating model parameter updates (line 16), and sending these aggregated updates to the FL server (line 17).

To compute the training assignments, the *Train* module calls the *CreateTrainingAssignments* module (lines 20-41) which implements the bin-packing heuristic solution to our problem. We begin by initializing the empty set of FedStarters and current maximum time bin capacity (lines 21-22). In order to run the first-fit decreasing bin-packing heuristic, we order the data chunks by decreasing size (line 23) and iterate over this ordered list (lines 24-40). For each data chunk, we firstly filter the time bins in order to ensure that we do not introduce an assignment that violates one of our energy, storage, or trust constraints (line 25). Next, we consider assigning the data chunk to every eligible time bin and compute the time bin's updated expected completion time (lines 26-30), based on equation 3. We proceed by selecting the fastest available time bin (line 31). We assign the data chunk from the FedSub to the FedStarter associated with this time bin, creating a new FedStarter-FedSub relation (lines 32-34) and updating the FedStarter's completion time and resources (lines 35-36). Finally, if this FedStarter is now the slowest FedPlayer, we update the current maximum time bin capacity (lines 37-39).

V. Evaluation

We evaluate the performance of FedTeams using Python simulation on a Tesla K80 GPU with 2.70 GHz. We compare FedTeams performance to the Standard Cloud-based FL [5] and Hierarchical Edge/Fog-based FL [18] baselines. We study the impact of varying different parameters; e.g. data distribution, levels of client heterogeneity, and degrees of trust on training performance. We start by presenting our experimental setup, followed by the results analysis.

A. Experimental Setup

1) Hardware and System Parameters: We simulate a university building with 4 floors and 5 classrooms on each floor, a realistic setting for FL training. Given average classroom size of 20 students and statistics on IoT usage [24], [25], we assume the following device distribution per classroom: 18 smartphones, 6 laptops, 3 tablets, and 3 smartwatches. For each device type, we identify the device model dominating today's market share and base average computation and energy consumption rates on this device's benchmarks [26], [27].

We sample individual values of client's computing power and energy consumption from a normal distribution with the averages listed below and a standard deviation that is 20% of the average. The average computational capacity of smartphones, laptops, tablets, and smartwatches is set to [26.3, 79.7, 28.5, and 1.24] GFlops; respectively [26], [27]. Additionally, we set average communication energy consumption rates of 1.25 Joules/Mb (Client-Client), 3.13 Joules/Mb (Client-Edge), and 9.38 Joules/Mb (Client-Cloud) [28].

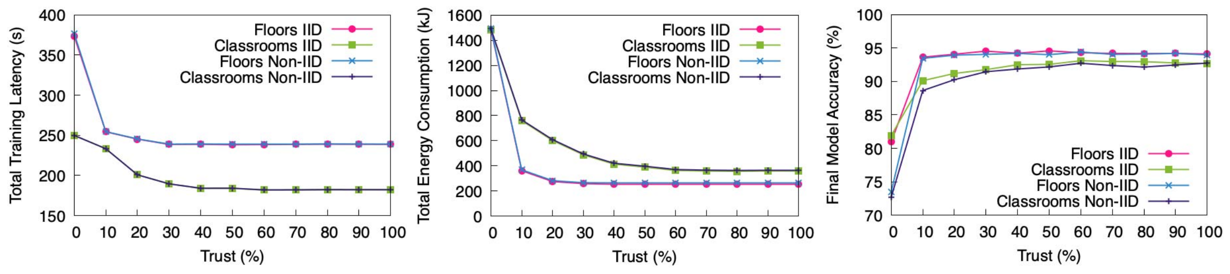
For communication, we assume the use of WiFi 802.11ac with maximum bandwidth of 450 Mbps on the 2.4 GHz band, and average latencies of 15 ms (Client-Edge), 85 ms (Edge-Cloud), and 100 ms (Client-Cloud). Given that all clients are sharing the communication channel, we assume a 25% performance drop due to network traffic and congestion, yielding average communication speeds of 13.11 Mbps (Client-Edge), 3.08 Mbps (Edge-Cloud), and 1.97 Mbps (Client-Cloud) [29], [30]. Past research has studied WiFi Direct indoors and measured speeds of 30.0 Mbps between devices within classrooms and 15.0 Mbps between devices across a floor [31].

2) Data: All FL approaches are tasked with training a Multilayer Perceptron (MLP) image classifier model. We build the neural network using the TensorFlow Federated (TFF) framework [32] and train on the MNIST dataset, which consists of 42,000 images of handwritten single digits between 0 and 9 (i.e. 10 data classes) [33]. We vary data distributions in order to create two types of clients: identically and independently distributed (IID) and non-identically and non-independently distributed (Non-IID) clients. IID clients are created by dividing the data equally across all clients and ensuring that each client has the same distribution of class labels, while Non-IID clients are created based on the NonIIDnessLevel metric. This metric defines the number of classes that appear at a single client, and in our experimentation we set it to 1 as we aim to showcase the difference between the IID setting and the contrasting highly Non-IID setting. The data in each class is divided equally across clients assigned to that class.

3) FL Baselines: The Standard FL [5] baseline is composed of 600 clients that all register, complete local training, and communicate model parameter updates to the Cloud-based FL Server at each round; for a total of 100 rounds. The Hierarchical FL [18] baseline includes the same 600 clients registering and training in all rounds, however split into 20 sets of 30 clients, with each set overseen by a single Edge Server. The Edge Server is responsible for aggregating model parameter updates from all clients under it. After 4 rounds of Edge aggregations, each Edge Server uploads its model to the Cloud-based FL Server for global aggregation. We run for 25 global FL rounds, resulting in a total of 100 training rounds.

For FedTeams, we use the same 600 clients, split into either 20 FedTeams of 30 FedPlayers (classroom setting) or 4 FedTeams of 150 FedPlayers (floor setting). Only the clients selected as FedStarters complete local training and communicate model parameter updates to the FedCaptain, whilst FedSubs offload data to their assigned FedStarters. For a fair comparison to Hierarchical, each FedTeam runs 4 rounds of aggregations between each global aggregation, 25 global FL rounds, and thus a total of 100 training rounds.

4) Metrics and Parameters: Table II summarizes our experimental parameters, their ranges, and nominal values.



(a) Total Training Latency (b) Total Energy Consumption (c) Final Model Accuracy

Fig. 2. Impact of varying the fraction of trusted relationships

TABLE II
EXPERIMENTAL PARAMETERS

Parameter	Values
FL Approach	{Standard, Hierarchical, FedTeams}
Trust %	{0, 10, 20, 30, ... , 90, 100}
Client Heterogeneity	{None, Low, Medium, High}
Data Distribution	{IID, Non-IID}

Throughout our experiments, we assess the system performance using the following metrics:

(Final) Model Accuracy: Accuracy achieved by the FL server's global model on its test data following aggregation of all client updates at the end of each FL round. Final Model Accuracy refers to this result at the end of all FL rounds.

Total Training Latency: Time elapsed between first distribution of initial global model from FL server and final global model aggregation at the end of final FL round. This includes latency incurred by sending/receiving model parameters, sending/receiving training data, and training the model.

Total Energy Consumption: Energy consumed by all clients in all FL rounds; for sending/receiving model parameters, sending/receiving training data, and training the model.

B. Results

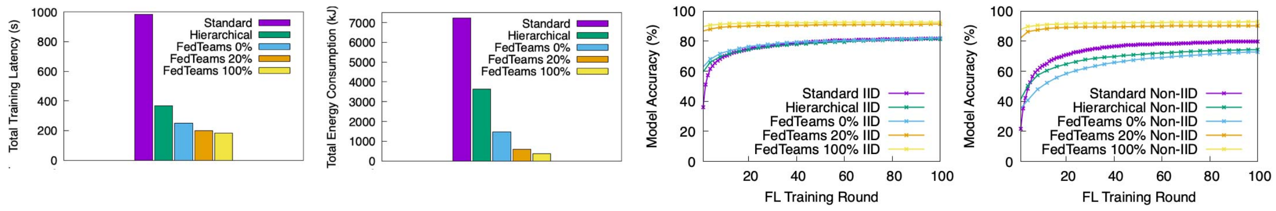
1) Impact of Trust on Different FedTeams Settings: In the first experiment, we inspect how varying the degree of trust relationships impacts training graph formation and training performance for different FedTeam settings. We vary trust by considering all $\binom{p}{2}$ client relations within a FedTeam of p FedPlayers, and randomly and progressively selecting 0%, 10%, 20%,...,100% of these relations to be trusted (i.e., the clients can offload data to each other). We consider two FedTeams settings: one where each classroom forms a FedTeam (20 FedTeams of 30 FedPlayers each) and one where each floor, containing 5 classrooms, constitutes a FedTeam (4 FedTeams of 150 FedPlayers each).

We observe that the training latency and energy results for each setting in Figure 2(a) and Figure 2(b) are identical across the IID and Non-IID settings. This indicates that data distribution does not impact training latency or energy consumption when the number of training rounds is fixed. We note that if we instead instructed FL training to continue until model convergence, the IID setting would achieve much lower latency and energy consumption due to faster global model convergence exhibited in Figure 2(c).

Figure 2(a) demonstrates classroom FedTeams achieve 24.4% lower training latency compared to floor FedTeams. This lower latency is due to FedPlayers leveraging closer-proximity communication within the same classroom as opposed to communicating across the distance and physical barrier of multiple classrooms. However, Figure 2(b) and Figure 2(c) show that floor FedTeams achieve 28.1% less energy consumption and a 1.4% increase in model accuracy relative to classroom FedTeams. This lower energy consumption and higher accuracy are a result of the fact that larger FedTeams allow for more data offloading. Powerful FedStarters are able to aggregate data from up to 149 FedSubs across the floor, whereas in classroom FedTeams the data can be aggregated from at most 29 FedSubs. Consequently, floor FedTeams are likely to have fewer FedStarters, reducing energy consumption as most clients become FedSubs that offload data once and don't expend energy on training or communication in consequent FL rounds. The aggregation of more data at fewer FedStarters also increases the final model accuracy as it brings training and performance closer to that of centralized ML. Given our goal of minimizing latency, we use the classroom setting in our remaining experiments.

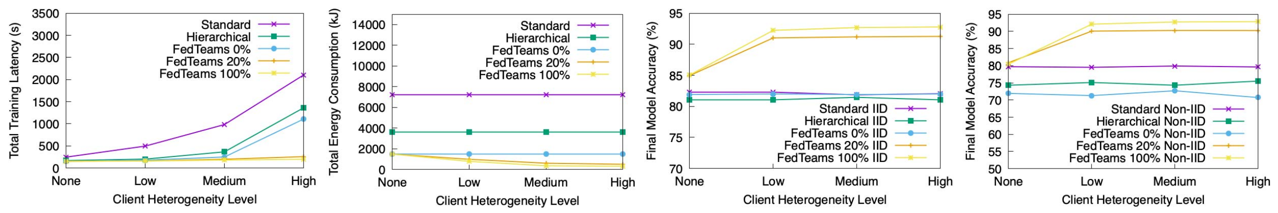
Focusing on the impact of trust across Figure 2(a)-(c), we compare the improvements achieved with only 20% trust to the maximum possible improvements (i.e., 100% trust). We observe that up to 94% of training latency reduction, 83% of energy savings, and 75% of model accuracy increase can be achieved with only 20% trust. This stresses that a little trust can go a long way, with FedTeams requiring trust between only a fraction of the FedPlayers to achieve major improvements.

2) Performance versus State-of-the-art FL: Figure 3(a) shows the relative performance between FedTeams and the baselines in terms of total training latency. We observe the expected latency reduction with Hierarchical FL compared to Standard FL, as the clients communicate with a nearby Edge Server instead of a distant Cloud Server at each round. Since no data offloading is possible in the FedTeams scenario with 0% trust, the reduction in latency between Hierarchical FL and *FedTeams*_0% is attributed to FedTeams exploiting closer proximity and thus lower latency communication. A further drop in training latency is achieved by *FedTeams*_20% due to the optimization algorithm's selection of resource-rich and highly-connected clients for training. Finally, we see a less abrupt decrease as we reach the FedTeams scenario with full



(a) Total Training Latency (b) Total Energy Consumption (c) Model Accuracy (IID) (d) Model Accuracy (Non-IID)

Fig. 3. FedTeams vs. Standard FL vs. Hierarchical FL



(a) Total Training Latency (b) Total Energy Consumption (c) Model Accuracy (IID) (d) Model Accuracy (Non-IID)

Fig. 4. Impact of varying the level of client heterogeneity

trust, altogether achieving improvements of 81.6% and 50.4% relative to Standard and Hierarchical FL, respectively.

Figure 3(b) shows FedTeams achieving energy savings of up to 95.0% and 90.1% relative to Standard and Hierarchical FL, respectively. There are two reasons for FedTeam's observed reduction in total energy consumption. First, in the baseline scenarios, all clients are consuming energy by running local training and communicating parameter updates at each round. In contrast, FedTeams allow a large fraction of clients (FedSubs) to offload data at the start of training and do no work in all remaining rounds. Thus, a FedSub's energy cost of offloading data once is extremely small relative to the energy spent on training and communication. Second, FedTeams leverage D2D communication which consumes less energy compared to cellular communication with a Cloud/Edge Servers.

Focusing on FedTeams in Figure 3(c), we note an improvement in final global model accuracy of 10.8% relative to Standard FL and 11.2% relative to Hierarchical FL. This is as a result of FedTeams aggregating data at the FedStarters, thus bringing model accuracy closer to that of centralized approaches. This gain of aggregating data increases when looking at Figure 3(d), with FedTeams achieving an accuracy 12.8% higher than Standard FL and 18.4% higher than Hierarchical FL. This increased margin is due to data aggregation resulting in greater data balance and thus counteracting the negative impact of Non-IID data. Hence, the clients train using data set distributions and optimization directions that are nearer to that of the global data set and model.

3) Impact of Client Heterogeneity: Lastly, we investigate the impact of client heterogeneity on the performance of FedTeams relative to baselines. We build 4 normal distributions with standard deviations of: 0% (None), 10% (Low), 20% (Medium), and 40% (High). We sample client computation and communication speeds from these distributions and independently of one another.

Figure 4(a) shows that as the level of client heterogeneity increases, the gap in training latency between the baselines and FedTeams (with non-zero trust) begins to grow rapidly. Comparing Standard FL and *FedTeams*_20%, the gap jumps from 783 seconds to 1,845 seconds between the medium and high heterogeneity settings. Looking at Hierarchical FL and *FedTeams*_20%, the gap jumps from 166 seconds to 1,104 seconds between the medium and high heterogeneity settings. This jump occurs because the high heterogeneity setting implies the presence of straggler clients with low communication and computation speeds, slowing down FL training in Standard and Hierarchical FL. However, FedTeams mitigates the detrimental impact of these stragglers by instructing them to offload their data to high-performing clients that execute the training and maintain low latency.

Since the average values of all normal distributions are the same, the energy consumption results in Figure 4(b) remain stable for the baselines and *FedTeams*_0%. As for FedTeams with non-zero trust, we note improvements of up to 47.2% between none-low, 28.6% between low-medium, and 3.4% between medium-high client heterogeneity. This is due to FedTeams selecting the fewer powerful clients as FedStarters in highly heterogeneous settings, and conversely selecting many clients as FedStarters in homogeneous settings as no time reduction is achieved by offloading data to an equally powerful and well-connected device. As in previous experiments, fewer FedStarters result in reduced energy consumption, yielding the observed drop as heterogeneity increases.

Similarly, Figure 4(c) shows the baselines and *FedTeams*_0% producing models of roughly equal accuracy across all heterogeneity levels. Meanwhile, as heterogeneity increases, *FedTeams*_20% displays accuracy improvements of 4.0%, 0.2%, and 0.1% and *FedTeams*_100% displays accuracy improvements of 7.2%, 0.5%, and 0.1%. In the Non-IID setting presented in Figure 4(d), *FedTeams*_20% achieves improvements of 7.2%, 0.2%, and 0.1%, and *FedTeams*_100% achieves improvements of 11.6%, 0.4%, and 0.2%. These results align with previous experiments, affirming that FedTeams offer greater accuracy improvements in Non-IID settings. As before, this is a result of FedTeams selecting fewer powerful FedStarters in highly heterogeneous settings, thereby aggregating more data and bringing training closer to a centralized approach that achieves high accuracy.

We emphasize that in reality, mobile/IoT client devices are highly heterogeneous. Our results highlight the struggle that Standard FL and Hierarchical FL face in such highly heterogeneous settings, with straggler clients slowing down the entire training. FedTeams is proven to be resilient to stragglers as the results show. FedTeams result in stable latency and improvements in accuracy and energy consumption as client heterogeneity increases. Altogether, this demonstrates that the potential gain achieved by using FedTeams is maximized in realistic FL scenarios where clients are highly heterogeneous.

VI. Conclusion and Future Work

In this paper, we built the FedTeams system that minimizes FL training latency by leveraging social trust to create a training assignment that optimizes training on powerful clients whilst including all available training data. We evaluated FedTeams to demonstrate up to a 81.6% decrease in FL training latency, 90.1-95.0% reduction in total energy consumption, and up to a 11.2% increase in global model accuracy. Future research directions include exploring different techniques for FedTeams formation, considering factors such as client network proximity, data distribution, and trust relations. The optimization objective could also be reformulated to focus on minimizing energy consumption or maximizing accuracy. Finally, the definition and mechanism of trust in the context of FL should be studied, considering non-binary trust and the potential impact of training on encrypted data.

Acknowledgement

This publication was made possible by PDRA 6-0624-20020 from the Qatar National Research Fund (a member of Qatar Foundation). The contents herein reflect the work, and are solely the responsibility, of the author(s).

References

[1] H. Abdelnasser, K. Harras, and M. Youssef, "A ubiquitous wifi-based fine-grained gesture recognition system," *IEEE Transactions on Mobile Computing*, vol. 18, no. 11, pp. 2474–2487, 2018.

[2] A. Essameldin, M. Nurulhoque, and K. A. Harras, "More than the sum of its things: Resource sharing across iots at the edge," in *ACM/IEEE SEC*, 2020.

[3] A. Saeed, M. Ammar, K. A. Harras, and E. Zegura, "Vision: The case for symbiosis in the internet of things," in *SIGCOMM MCC Workshop*. ACM, 2015, pp. 23–27.

[4] M. A. Shah, K. A. Harras, and B. Raj, "Sherlock: A crowd-sourced system for automatic tagging of indoor floor plans," in *IEEE MASS*, 2020.

[5] B. McMahan, E. Moore, D. Ramage, and et al., "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273–1282.

[6] 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.

[7] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. Vincent Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys Tutorials*, vol. 23, no. 3, pp. 1622–1658, 2021.

[8] R. Saha, S. Misra, and P. K. Deb, "Fogfl: Fog-assisted federated learning for resource-constrained iot devices," *IEEE Internet of Things Journal*, vol. 8, no. 10, pp. 8456–8463, 2020.

[9] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, pp. 1–7.

[10] E. Diao, J. Ding, and V. Tarokh, "Heterofl: Computation and communication efficient federated learning for heterogeneous clients," 2021.

[11] Z. Chen, D. Li, M. Zhao, S. Zhang, and J. Zhu, "Semi-federated learning," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2020, pp. 1–6.

[12] M. Duan, D. Liu, and X. e. a. Ji, "Fedgroup: Efficient federated learning via decomposed similarity-based clustering."

[13] Z. Yang, M. Chen, W. Saad, C. S. Hong, M. Shikh-Bahaei, H. V. Poor, and S. Cui, "Delay minimization for federated learning over wireless communication networks," 2020.

[14] S. Samarakoon, M. Bennis, and W. e. a. Saad, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1146–1159, 2020.

[15] S. Wang, Y. Ruan, Y. Tu, S. Wagle, C. G. Brinton, and C. Joe-Wong, "Network-aware optimization of distributed learning for fog computing," 2020. [Online]. Available: https://arxiv.org/abs/2004.08488

[16] G. Shang-Fu and Z. Jian-Lei, "A survey of reputation and trust mechanism in peer-to-peer network," in *2012 International Conference on Industrial Control and Electronics Engineering*, 2012, pp. 116–119.

[17] A. Mtibaa and K. A. Harras, "Social-based trust in mobile opportunistic networks," in *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, 2011, pp. 1–6.

[18] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[19] H. Garcia-Molina, "Elections in a distributed computing system," *IEEE transactions on Computers*, vol. 31, no. 01, pp. 48–59, 1982.

[20] C. Hoffman, "Mobile cpus are now as fast as most desktop pcs," https://www.howtogeek.com/393139/mobile-cpus-are-now-as-fast-as-your-desktop-pc/, 2018.

[21] K. Parrish, "Arm's future cpu designs may finally catch up with intel in laptops by 2020," https://www.digitaltrends.com/computing/arm-cpu-roadmap-reveals-laptop-takeover-plan/, 2018.

[22] T. Tuor, S. Wang, B. J. Ko, C. Liu, and K. K. Leung, "Overcoming noisy and irrelevant data in federated learning," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 5020–5027.

[23] E. W. Weisstein, "Bin-packing problem," http://mathworld.wolfram.com/Bin-PackingProblem.html, 2000.

[24] "Average class size in public schools, by class type and state," U.S. Department of Education, National Center for Education Statistics, National Teacher and Principal Survey (NTPS), Tech. Rep., 2018.

[25] J. Gaubys, "Most popular devices in the us," https://www.oberlo.com/statistics/most-popular-devices, 2020.

[26] "Geekbench browser," https://browser.geekbench.com/v5/compute, 2022.

[27] H. Gedawy, K. A. Harras, K. Habak, and M. Hamdi, "Femtoclouds beyond the edge: The overlooked data centers," *IEEE Internet of Things Magazine*, vol. 3, no. 1, pp. 44–49, 2020.

[28] A. Mtibaa, K. A. Harras, and A. Fahim, "Towards computational offloading in mobile device clouds," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 1, 2013, pp. 331–338.

[29] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the internet of things: A case study," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, 2018.

[30] SWITCH, "Tcp throughput calculator," https://www.switch.ch/network/tools/tcp_throughput/.

[31] A. Mtibaa, A. Emam, S. Tariq, A. Essameldin, and K. A. Harras, "On practical device-to-device wireless communication: A measurement driven study," in *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, 2017, pp. 409–414.

[32] M. Abadi, A. Agarwal, P. Barham, and et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[33] 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.